Introduction to Problem Solving and Programming

Prof. Deepak Gupta

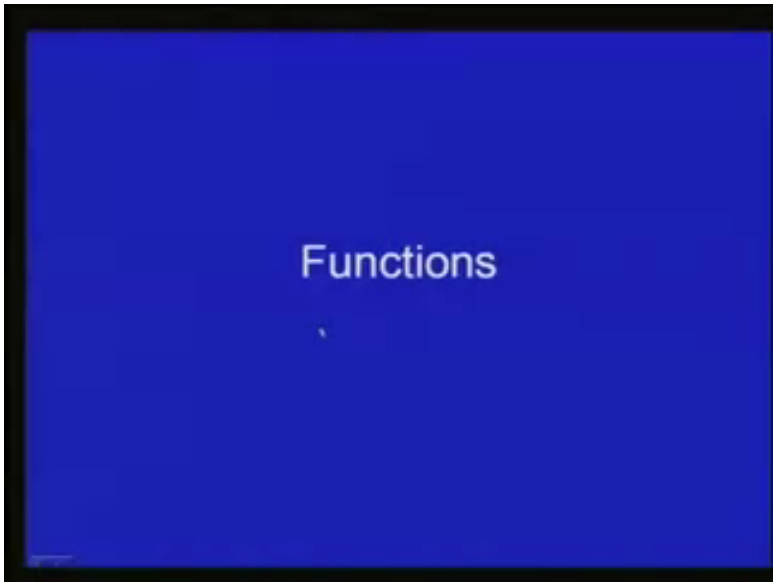
Department of Computer Science Engineering

Indian Institute of Technology, Kanpur

Lecture - 14

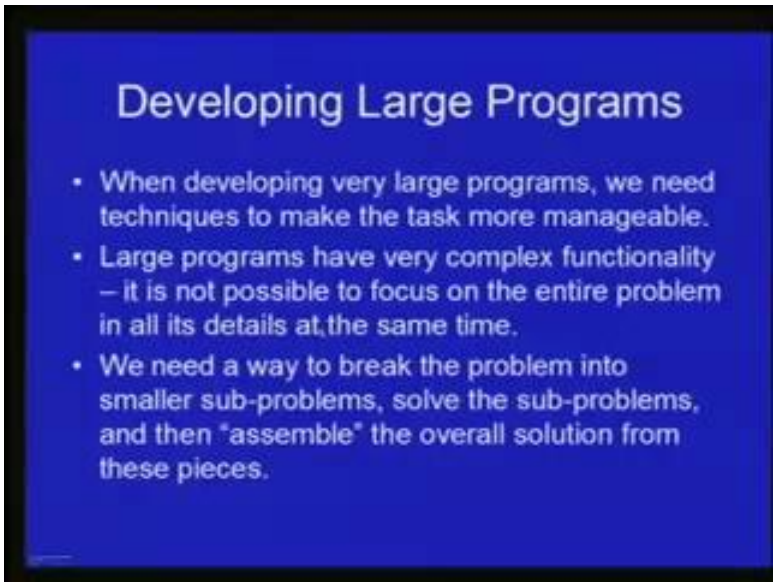
Today we will discuss a very important concept in programming and that is the concept of breaking up a large problem into smaller sub problems, and then we able to write independent small program segments for solving these problems and then we being later able to assemble them to form the solution to the overall problem.

(Refer Slide Time: 00:47)



In C, this is achieved by using what are known as function, we have already used various library functions in our programs that we have written so far. Today, we will see how we can write and define our own function, which can be used for simplifying the problem by breaking it up into small problems and so on.

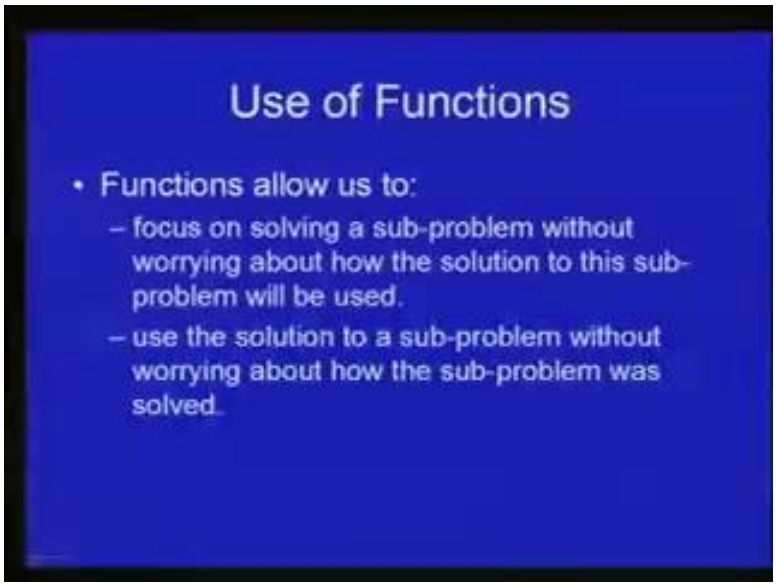
(Refer Slide Time: 01:02)



So, the problem is that in contrast to the small programs that we have been writing so far. When one is developing very large programs then we need to make the task of programming much more manageable. Large programs are needed, because we know many real life problems are very very complex and obviously, they require complex solution but the capabilities of the human mind are limited. So, the human mind cannot really focus on all the details of a very complex problem all at the same time. And therefore, to ease the process of programming or developing a program, what one needs to do is to divide up the problem into smaller parts, and then try to solve these smaller problems individually and later on assemble the solution to the overall problem for from the solution of these individual smaller problems.

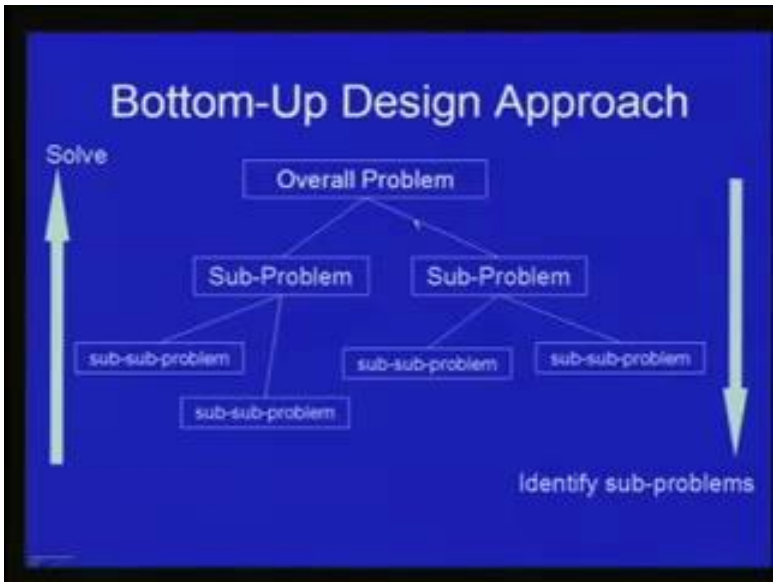
Of course, it may turn out that even the smallest of problems that we identify are still very complex, in that case we need to further break them up into smaller sub problems yet and then try to solve them at that level. And this processes has to continue till the problems that we have reached are sufficiently simple that we can focus on solving one problem at a time.

(Refer Slide Time: 02:22)



So, this entire process in C is accomplished using what are known as function and essentially functions allows us to do the kind of breaking up of problems into smaller problems, and then assembling the overall solution from solution to the smaller problem. In the way, that we been talking about using function we can focus on solving a sub problem without really worrying too much about how the solution to the problem will be used in the solution to the overall problem. And when we are solving or assembling the solution to the sub problems to form the solution to the entire problem, you should not have to worry about how the individual sub problems were actually solved. So, both these kind of abstractions the function allow us to perform.

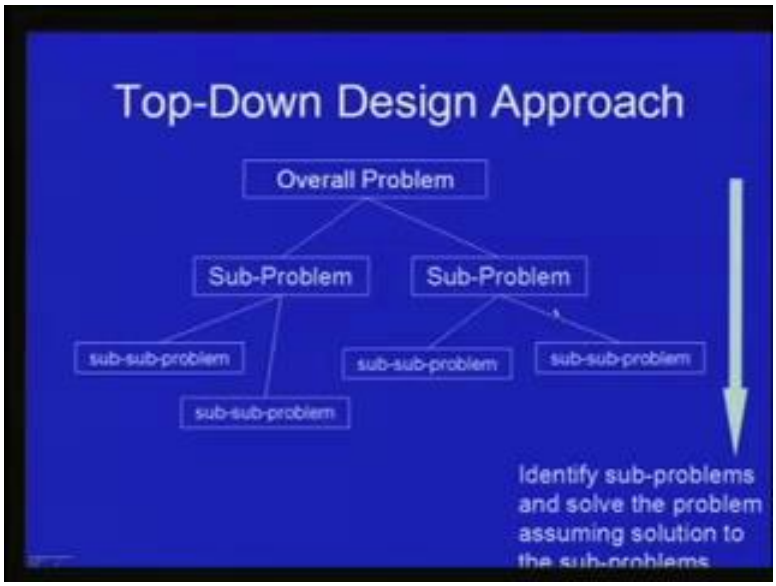
(Refer Slide Time: 03:13)



So, there are two using using this idea of breaking up the problem into smaller problems and so on, there are two major ways in which one can design solution or programs for a given problem, and these are fairly intuitive in their nature. So, let us look at one of them first that is the bottom up design approach. So, here essentially what you do is that given a problem, we identify the sub problem. So, let us say we have identified these two sub problems and then as I said these sub problems may themselves be still quite complex. So, we do this breaking up still further. So, maybe we have divided both these sub problems into two sub sub problems each, and now supposing that the smallest problem that we have identified are small enough or simple enough that we can solve each of it in its entirety.

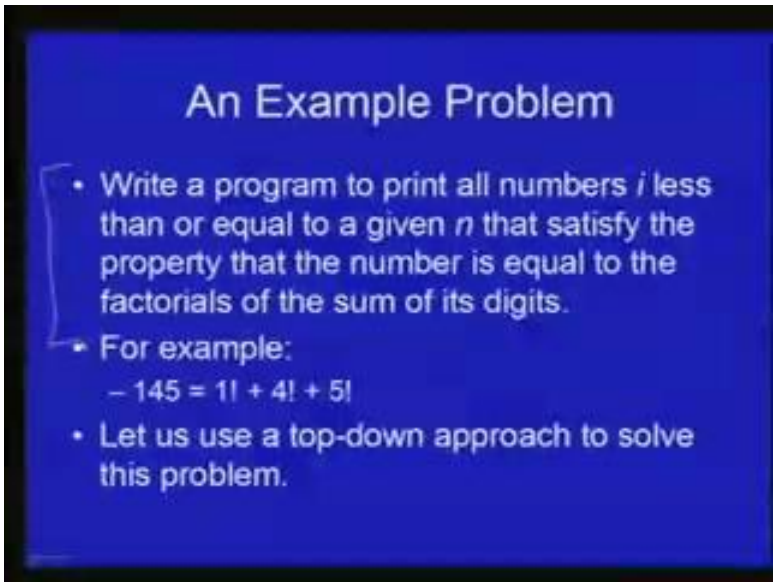
We can think about each of this sub problems in its entirety then we start up start solving these problem bottom up. So, we might solve this problem first and then this problem and then using the solutions to these sub problem. We might assemble the solution to speak of this problem. Similarly then we will solve this problem and this problem. And then assemble solution to this problem and having finally solved these two sub problems we assemble the solution to the overall problem. So, the solution the problems are solved in the upward direction while the identification is in the downwards direction.

(Refer Slide Time: 04:57)



In contrast to that, one can think about a top down approach where the procedure is like slightly different. So, again from the overall problem we identify sub problems and not just identify sub problems, assume at a certain level that the sub problems have already being solved and then assuming that solution to the sub problem exist, try to solve the overall problem first. So, here let us say we identify two sub problems, and without having solved this problem yet at this level we assume that the sub problems have already been solved, and design a solution to this problem assuming a solution for these smaller sub problem. And then of course, in the next step, we need to solve both of these sub problems. So, if they are still complex enough we again break them up into smaller parts. And now these sub problems would be solved assuming that these two smaller sub problems have already been solved and so on and so forth. So, that is the top down approach.

(Refer Slide Time: 05:53)



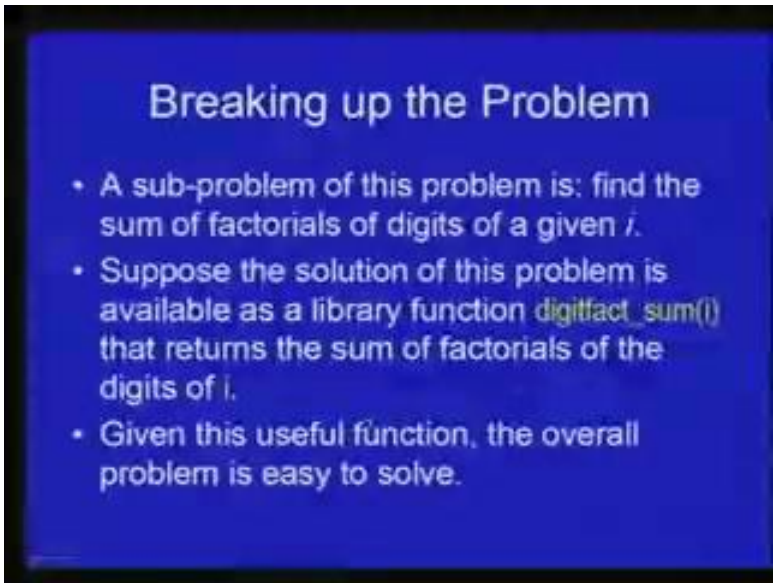
Let us now take a somewhat more complex programs then the ones we have been looking at so far to illustrate the ideas that we have been talking about. So, we will use the top down approach to solve this particular problem, which is actually still not very very complex, but it will have to illustrate the points that we are trying to make. So, in this problem, we are supposed to write a program that given an input integer n which is greater than one greater than or equal to one, the program is suppose to print all numbers i which are less than equal to n such that i equal to the sum of the factorials of its digits.

So, let us now take a simple example problem to illustrate the design approach that we are talking about. We will use the top down design approach in the solution to this problem. This problem is still not very complex, but it does have to illustrate the points that we have been trying to make and will also illustrate how we can define functions in our program and use them and so on and so forth.

So, the problem is very simple we are supposed to write a program which takes an input integer n as the input and n is supposed to be more than equal to one. And the program is required to print all the integers i which are less than or equal to n such that i is equal to the sum of the factorial of its digits. So, for example, one forty five is equal to one factorial plus four factorial plus five factorial you can easily verify that five factorial is one twenty, four factorial is twenty four, and one factorial is of course is one. So, one factorial plus four factorial plus five factorial is equal to one forty five. So, the program

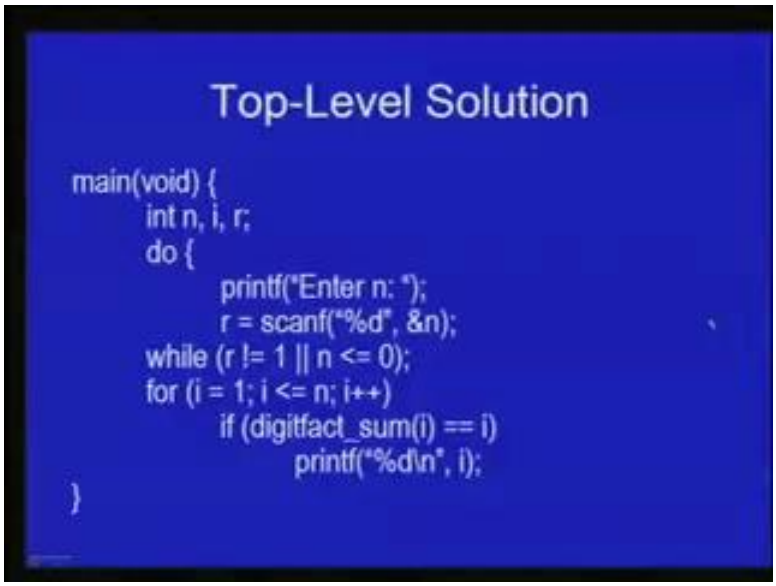
is suppose to print all such numbers less than equal to n and n is the taken as an input. So, let us use a top down approach to solve this problem, and to do that we need to first pick up the problem and identify smaller or simpler sub problem.

(Refer Slide Time: 07:53)



So, the sub problem of this problem that we can immediately identify is given an integer i find the sum of factorial of its digits. Now, in the top down approach instead of going and solving this sub problem first, lets us assume that the solution to this problem is already available to us. So, let us assume that a library function digit fact sum is already available which given an integer i returns the sum of the factorials of the digits of i. Of course, as we know, there is no such library function will have to write that our self and that will constitute the solution of these sub problem that we have identified, but at this stage we assume that this solution is available to us. In other words, there is already one such function that is available to us and all we have to do is to use this function in finding the solution to the overall problem and you can see that given this useful function the overall problem is quite easy to solve.

(Refer Slide Time: 08:58)



Here is the main what the main program might look like. We have declared three variable n i and r; n will be the input integer that will read, and i will be used to run a loop from one to n to check which number satisfy the given property, the variable r will used to hold the return value of scanf which we discussed last time. If we recall, we said that scanf actually returns the number of items successfully assigned. So, we will use and we should use this return value to check that whatever kind of inputs we were actually expecting the user did actually give that kind of input.

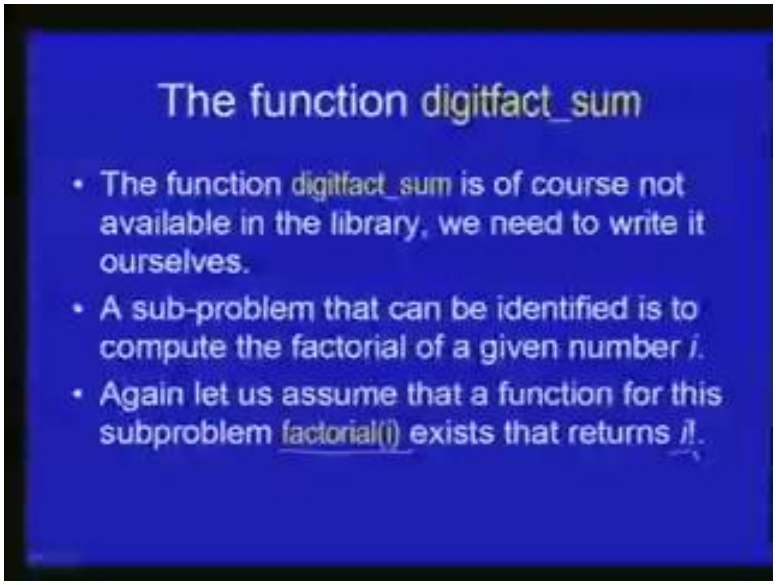
So, here is a familiar do while loop to read an input integer n and we want to make sure that the input integer n is at least one. So as long as n is less than equal to zero, we give the prompt to the user and then read the value of n, and r is return value of scanf. Note that the value of r will be other than one, if the input was not proper that is something other than digits and the plus minus sign etcetera was entered. So, if r is not one, that means, the user did not give a proper input and therefore, we should ask him to give the input again.

So, therefore this while loop is running as long as r is not equal to one which implies that improper input was given by the user or n is less than equal to zero, which means that some integer was entered, but it was less than equal to zero. Whereas we want a positive integer which is at least once and therefore, the user is prompted again. But that is the simpler part this is the main part of the program. All we need to do is to run loop from i

equal to one to i equal to n and for each i if the sum of the factorial of digits of i is equal to i then we print that i.

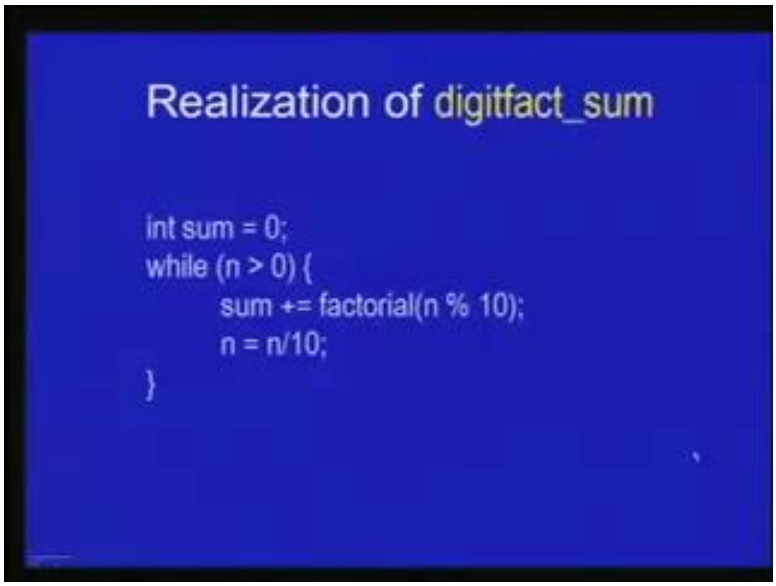
Note that we are assuming that this function digit fact sum i is already available to us, and we are simply using that function without worrying about how the sum of the factorials of the digits of this number i is being computed. Note also the call to this function is exactly the same as we have been calling the library functions so far, we just have to give the name of the function and within brackets we have to give the arguments for this function. So that is not the end of the program development because the function digit fact sum is not there in the library. So, we have to write that our self and that is the next step in the development of the program.

(Refer Slide Time: 12:06)



So, how do we solve the problem of finding the sum of the factorial of the digits of the given number while you can immediately see that it has a simpler sub problem and that is the finding out the factorial of the given number i. So, suppose we have the solution to this problem again in the form of a function which given finds i factorial. So, let us assume if factorial i function exist, which for a given value of i returns i factorial and given this function finding out the sum of the factorial of the digits is quite straightforward.

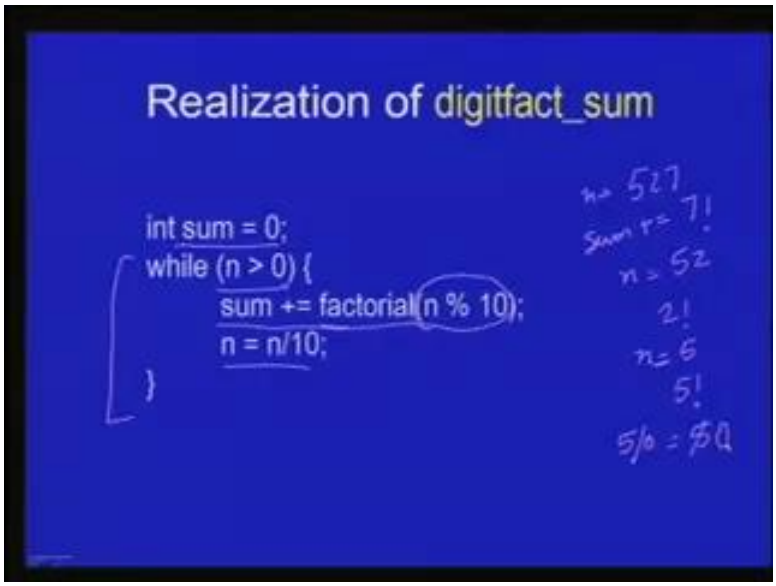
(Refer Slide Time: 12:19)



So, this is what the code for doing that might look like. So, lets us initialize the variable sum to zero which is going to hold the sum of the factorials of the digits of the number n. Let us assume that n is number for which we have to compute the sum of the factorial. So, what we are doing here is that as long as the number is greater than zero we find the least significant digit of n and that is easily obtained by taking the reminder after dividing n by n by ten and that is given by the expression n percent n. So, this n percent n represents the last digit or the least significant digit of the number n and. So, we take the factorial of that note that we are assuming that the factorial function already exist.

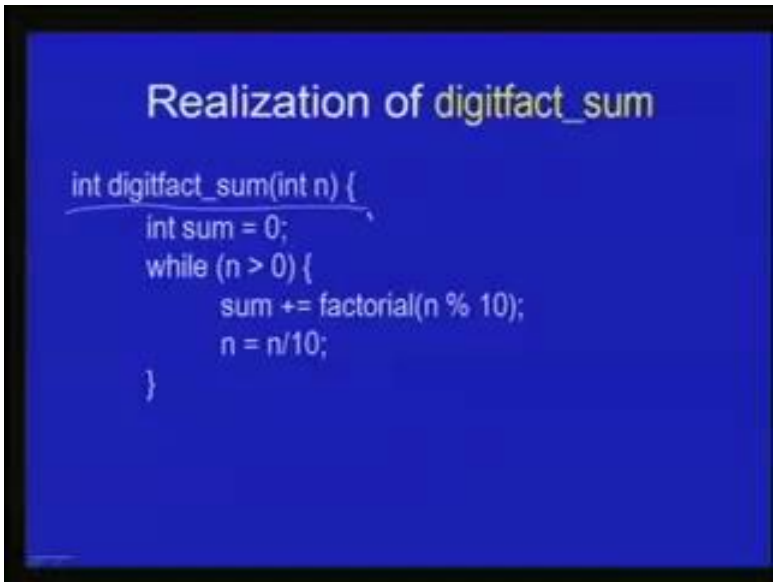
And whatever is the answer we add that to sum and after that we divide n by ten and remember this is integer division, because n is an integer and ten is also an integer. So, what this will do is that n will become the same number as previously accept that the last digit will get removed. So, for example, if n was 527 then in the first iteration n percent n would be seven and will add seven factorial to sum and then sum and then n will become n by 10 which is 52.

(Refer Slide Time: 14:26)



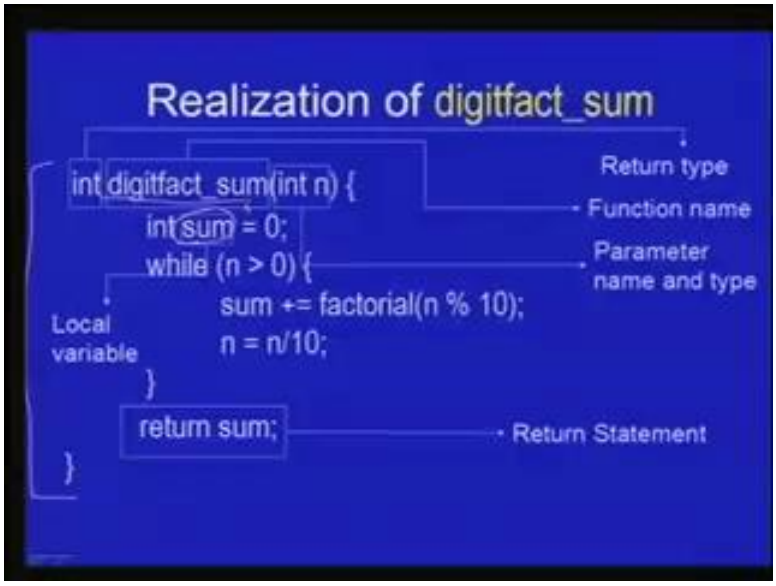
So, in the next iteration two factorial become added to sum and n will become five in the next iteration five factorial will be added to sum and n will become five by zero which is equal to five which is equal am sorry zero. So, and when that happens this loop will terminate because this will run as long as n is greater than zero and n has become zero as matter of fact. So, the write writing code is simple enough. And it requires only the knowledge of principles that you have already seen, but now what we need to do next is to wrap this code up and make it into a function. So, to do that we need to add some more stuff to this already existing code. So, this particular thing says that this is the function that we are defining this is the function header as it is called and this has three component.

(Refer Slide Time: 14:45)



The first component is the name of the function which is the digit fact sum the second component is the type of the return value that is what kind of value does this function result in and in this case that is an integer. So, the return type is an int and the last part is what is the name of the parameter and its type. So, this function takes one parameter which is being called n and its type is int and the entire code for the function is enclosed in braces.

(Refer Slide Time: 17:00)

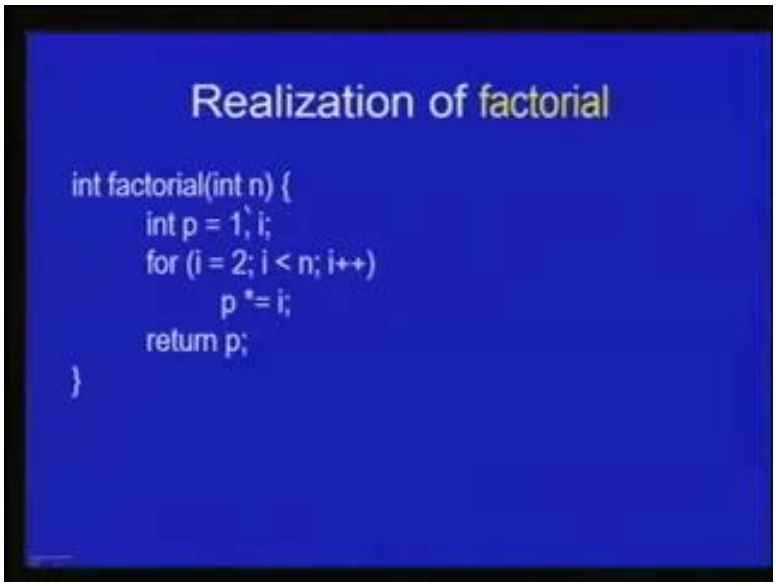


And of course, this brace has to close somewhere. So, we will do that next, but after the sum has been computed we need to somehow face that this sum that we have computed is. In fact, the result of evaluating this function or in other word this is the return value of this function. So, for doing that we need to add a return statement this is the return statement and this has two function first is that the expression that is given along with the return statement in this case which is simply the variable sum the value of this expression is the return value or the result of the evaluating this function.

So, in this case we have computed the result in the variable sum and. So, we are saying that the result to be returned to the caller whoever has who whichever function has called this function. The result that this function will be the value of the variable sum that is what we are saying here and the other thing that returns statement does is that the whenever the return statement is executed the function immediately terminates the execution of this function immediately terminates. And control goes to wherever the function was called from we will look at that in more detail with the help of some examples in a later slide.

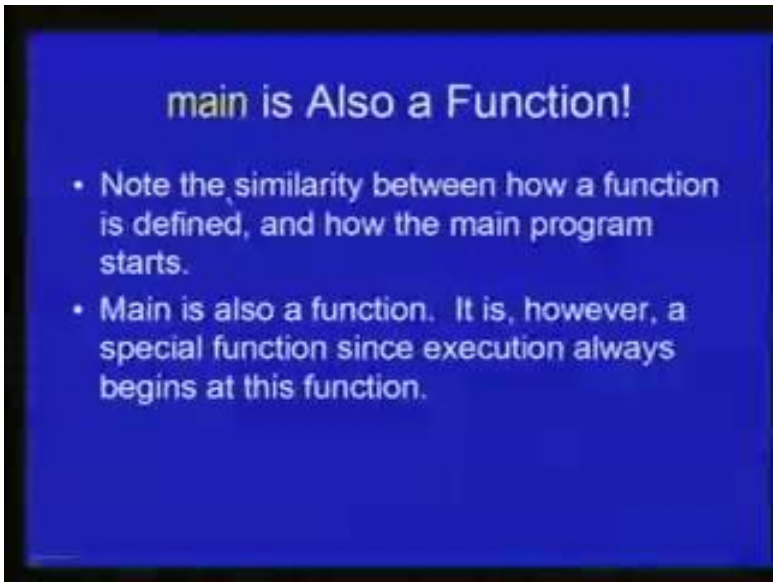
And finally, this variable sum which we have declared within the definition of this function is called a local variable of the function that we just defined which is digit fact sum? So, having returned the digit fact sum function we need to now write the factorial function this this is surely a simple function that you already know about and using what we have learnt now this is what the definition of the factorial function will look like.

(Refer Slide Time: 17:20)



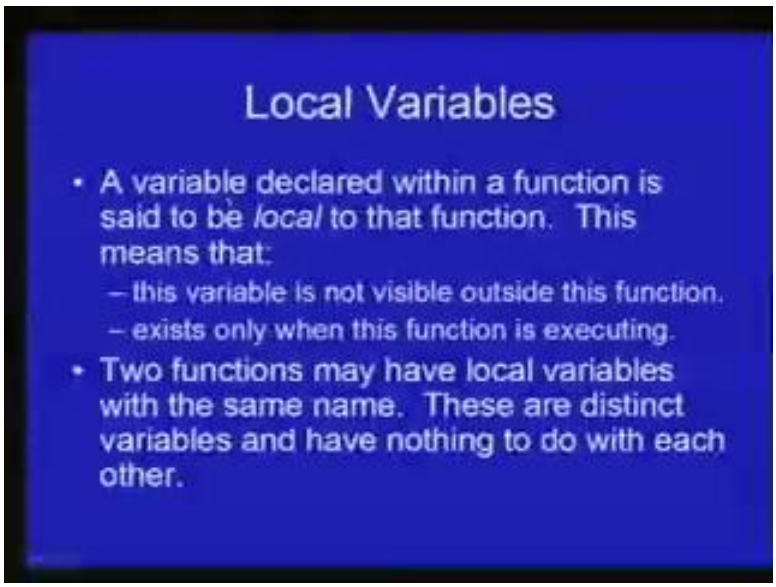
This code is very familiar to you from the computation of the factorial which we have been taking as an example in numerous programs. So, this this loop essentially multiplies number from two to up to n this should be i less than equal to n. And the result is stored in the variable p and the p is the final result of the evaluating this function and that is the return value again the name of the function is factorial the return type of the function is int. It has one argument which is called n and whose type is an int. So, you might be surprised to know or maybe not so surprised that the main program that we write the main is also a function. You might have noticed the similarity between how the main code starts and how a function is defined and that similarity is not coincidental it is because of the fact that main is also the name of a function.

(Refer Slide Time: 18:19)



Except its it is the same as any other function the only difference is that it is a special function in the sense that execution always start at this function. Let us now talk a little bit about local variable. So, we have already seen in the functions digit fact i digit fact sum and the function factorial that we can declare variables within the within the body of a particular function and these variables are said to be local to that particular function.

(Refer Slide Time: 18:46)



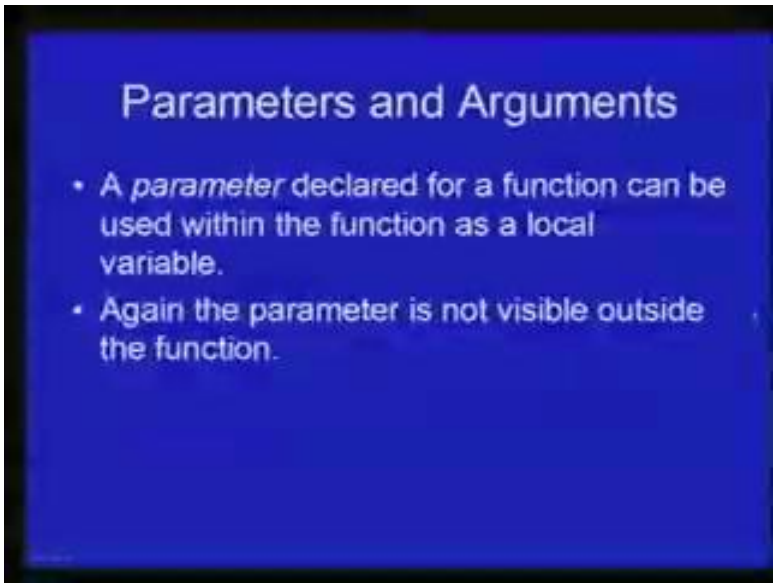
So, what local means is that the function declared within a particular function is not visible outside this function that is known as the scope of the function which will talk

about more in detail in the next lecture. And the second important point about local variable is that this variable has existence only while this particular function of which this is local variable is executing. So, while this function is not executing nobody has called this function and this variable does not exist at all there is no space allocated for this variable in the memory of the machine.

So, this idea is called the life time of a variable we will talk about life time of variable also in next lecture what is important to note is that it is possible that two function have local variables which have the same name. For example, if function factorial had a variable called i and the function main also had a local variable i now these two i's have nothing to do with each other the i declared within the function factorial is local to the function factorial.

And similarly, the i declared within the function main is local to the function main. So, within main, whenever we refer to the variable i what is meant is a reference to the variable i which is declared within the function main. Whereas within the function factorial when we refer to the variable i what is meant is a reference to the variable i declared within the function factorial.

(Refer Slide Time: 20:37)

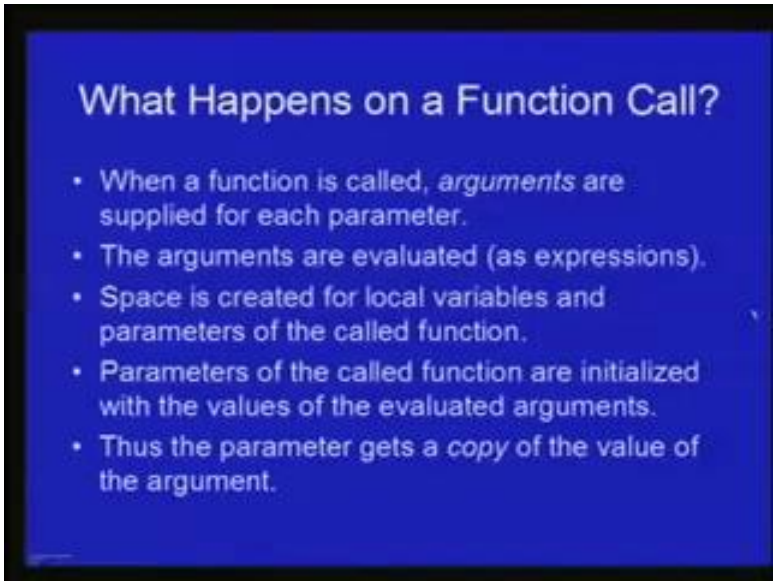


Let us now look at parameters and arguments. So, we saw that when we declare a function we have to declare what its parameters are we have seen so far. Only function with one parameter, but in general a function may have more than one parameter. Each

of those parameter in the function definition has to be given a name for example, in the factorial function the parameter were called n and it has to be given a type in the case of factorial of function that was the type was int.

When we call a function at that time an actual argument for that function has been supplied we will talk about the relationship between parameters and arguments shortly. But the important point to understand at this point in time is that a parameter declared for a function can be used pretty much like a local variable of that function. It behaves like a local variable that in terms of both scope and life time it is similar to local variables what that mean is that this variable is not visible outside this function and comes into existence only when that function gets called. So, for example, in the program that we just developed the factorial function had a parameter called n and the main function had a local variable called n now these two n's are again completely independent of each other and have nothing to do with each other.

(Refer Slide Time: 23:18)

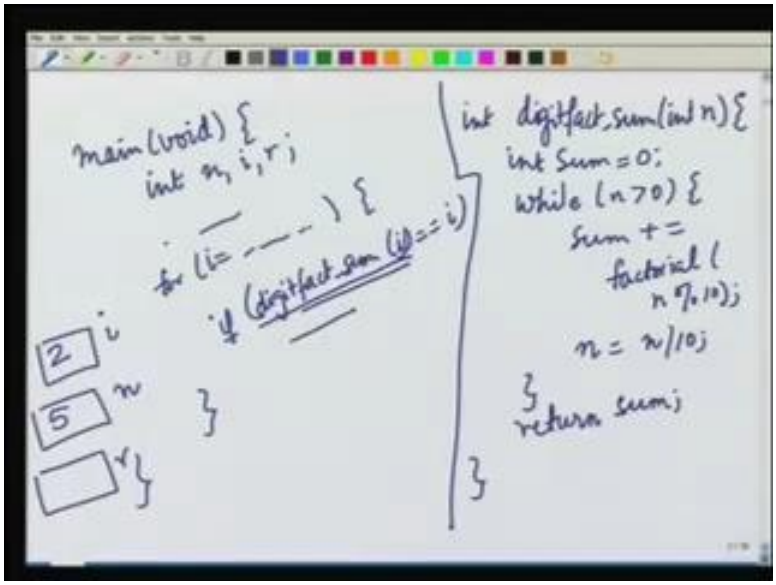


Let us now try to understand what happens when a function is called. So, when a function is called some parameters are supplied some arguments are supplied for that function. So, the arguments are supplied for each parameters that the function has if the function has two parameters then there must be two arguments if it has three parameters then there must be three arguments. What happens first is that these arguments are evaluated these arguments have to be expression and the evaluation of these expression

results in some value and then for the function which is being called the compiler creates space in memory for the local variables and parameter of this function.

And next step is that the parameter the space that has been created for the parameters of this function those parameters are initialized or their given values which are equal to the values of the corresponding argument expression. We will see that as an example and the important point to note is that the value that the parameter gets is a copy of the value of the corresponding argument. So, let us try to understand what is happening when a function is being called and some argument are being passed to it with the help of an example. I have scribbled here the definitions of the digit fact sum and the main function that we just wrote.

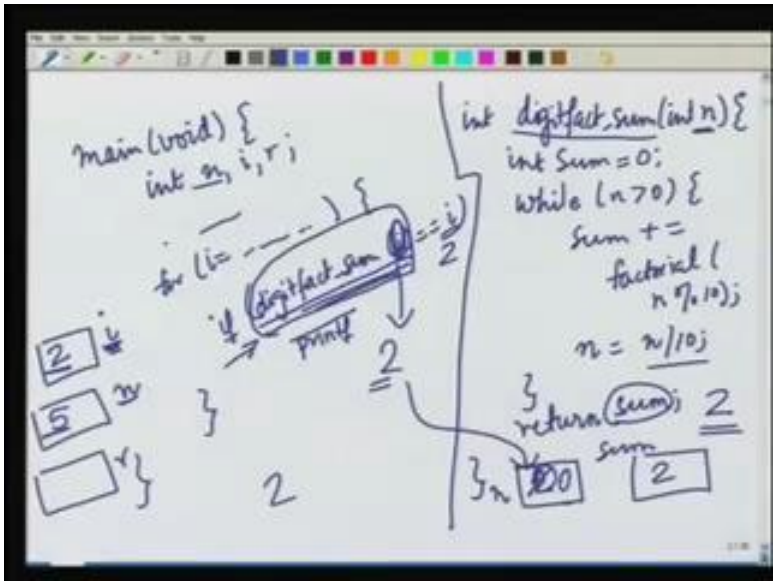
(Refer Slide Time: 24:28)



So, let us focus on this call from the main function to the digit fact sum function So, while the main function is executing the variable the local variables of main are in existence. So, let us say this is this is the space created for these local variables n i and r let us assume that at some point in time n happens to be five and maybe i happens to be two. Now, the function digit fact sum is not executing right now and. So, therefore, the variables of the function digit fact sum do not exist at this point in time. Now, when this function is called from this statement the argument passed to this function is the expression i. So, what what is going to happen is that this expression i will be evaluated and that of course, results in the value two.

And now since this function digit fact sum is being called space is created for the variables and parameters of this function. So, there is one parameter n and there is one local variable sum. Note that n is also a local variable of the function main, but this n and this n have nothing to do with each other. So, this box is also called n because the name of this parameter is also n.

(Refer Slide Time: 26:55)



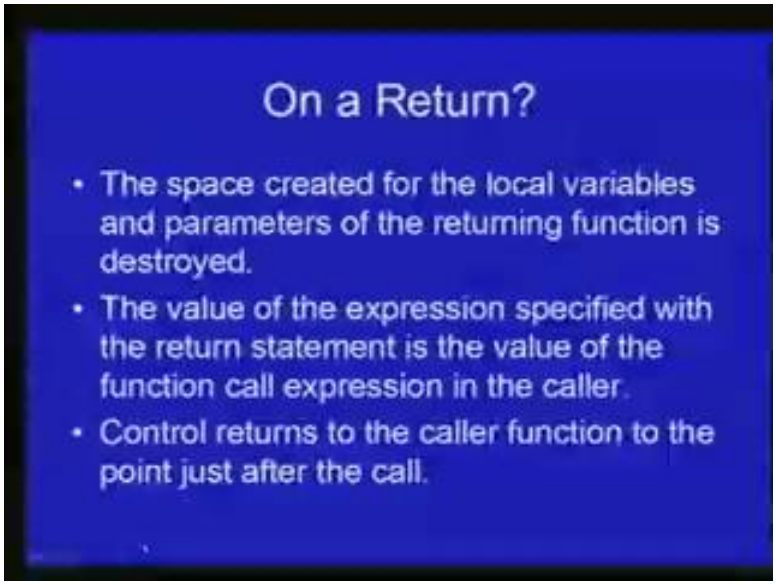
So, what is going to happen is that the value of this expression which has resulted in the value two is going to be copied into the value of n over here. And now the digit fact function will start executing and finally, as you will as you know the value of sum will become two and the value of n becomes zero because in every iteration we are dividing n by ten. So, in the first iteration itself n will become zero. So, the value of n is zero note that the value of this n which is the parameter of the function digit fact sum which has become zero. The value of i in the main function remains two that does not get changed and similarly the value of n in the main function that also does not get changed and it remains five.

Now, when this function is about to return this expression, which is the expression corresponding to the return value is the evaluated and the value of that expression is two. So, the value of this expression is what will become the value of this entire expression in the main function. So, when this function returns the control goes back to the main function and execution within the main function continues from the point just after where

the digit fact sum function was called. So, this function the main function now continues and the value of this entire expression comes out to be two which is the return value of the digit fact sum function and then of course, we check whether it is equal to i or not. The value of i is also two and so since these are equal the printf happens and to get printed on the output and so on.

So, what we just saw is that when a function is called from another function, the argument for each parameter have to be supplied, these arguments are evaluated and result in some value. Then the next step is to create space for each of the parameters and the local variables of the function, which is being called and the values of the argument are copied into the respective parameters and then the function body executes.

(Refer Slide Time: 27:26)



And finally, at the end, it returns a value and the value is taken as the value of the call expression that is the call to the function is treated as an expression. And the value of that expression is nothing but the return value that the function returns, and the execution starts in the function from which the function had been called at the point just after where the call was made. So, this is the end of this lecture; and in the next lecture, we will continue our discussion on functions and look at some more details involved with writing of function.