Programming and Data Structures
Prof. N. S. Narayanaswamy
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 03
Recursion

Hello, hi welcome to lecture number 3 of the first week of this course on programming data structures offered by NPTEL. So, to answer some of you questions on the forum in a brief manner. So, many of you a very concerned about the activities and the weightage for the activities, the course is going to be designed in such a way that the activities are for you to access, how well you learnt the material. And some of these activities will also challenge you and the sense that they will involve you to meticulously perform a certain amount to work after understanding the question.

For example, if you recall the insertion sort, comparison question many of few actually attempted to count the number of comparisons made by the algorithm insertion sort. Whereas, the question very clearly stated that you should count the total number of compressions made by the function insertion sort, so these are the certain things that you should make a note of when you perform these activities.

At the end of this lecture, you will also notice that the first programming assignment would have been released and the details of the submission dead line for this programming assignment will also be made a variable to you at the course web page itself. The other class of question have how to do with pointers, uniformly many of you have questions about the connection between pointers and arrays and why in scanf, ampersand is there and ampersand is not there.

The best way to sort these questions is to simply write a small program, here is a small recommendation for such a program defined an array and read elements into the array, using scanf. Now, what you try and do is, make the mistakes compile, run and see what kind of error messages you get and what kind of warning messages you get during compilation, this is the best way to actually understand. Of course, along with the standard text book that will help you, which will standard text book will guide you through your efforts.

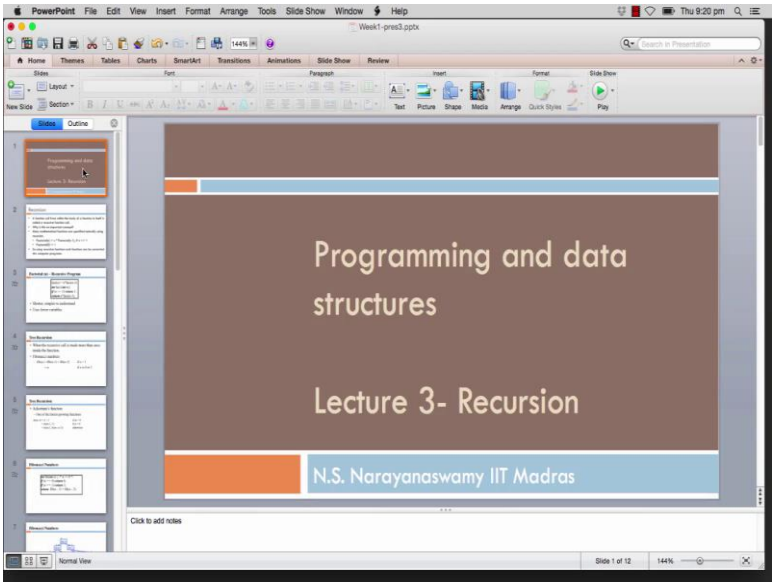
So, understand that universally when you read something using scanf, the format specifier tells you what data type is going to be a read from the input. Now, the question is, where should you store the value that has been read. And that information is given as a second argument to scanf, the first argument is a format specifier, the second argument tells you where to store that value.

Now, if you think of it this way understand that it is an address. And then, if you think of the name of an array is actually a pointer, then it already is an address and therefore, you will do not need an ampersand. So, work these details out and thinks would become clear, the best way is to actually write a small piece of code rather than, post on the forum and try to elicit answers. I hope most of you, when you post of forum you have already tried to an experiment with a small piece of code, before you post on the forum.

And today's lecture, we move one more step towards studying data structures, we study the concept of recursion. And recursion is extremely important, because many methods that you write now, when you design a data structure. Recall designing a data structure is to define an appropriate data type and talk about how or write functions which will tell you how this data structure is going to grow and shrink, many of these functions are best specified as recursive functions.

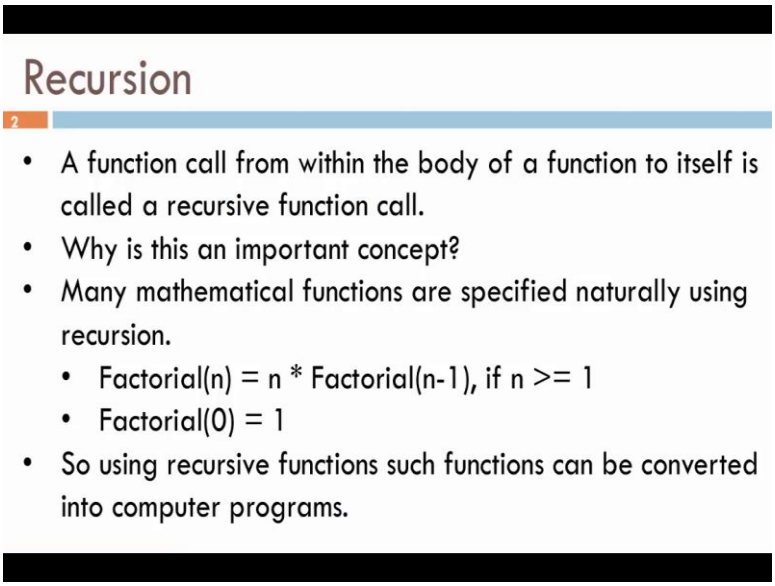
Therefore, a certain level or a significant level of a familiarity with recursion is extremely important. And in today's lecture, we will look at simple recursively specified functions then, we will see a program in which the function definition is recursive and we will see how it works. So, today's lecture is likely to be relatively shorter than the previous lectures. So, now let us go to the slides for today's lecture.

(Refer Slide Time: 05:11)



So, today's lecture is about Recursion it is a 3rd lecture in the first week of this course.

(Refer Slide Time: 05:23)



So, let us just quickly get past the definition of a recursive function. So, a recursive function is first of all a function and it is called recursive, because from the body of the function definition, a function call is met, so such a function is a recursive function. Now, why this is an important concept, recursive itself is very important, because many mathematical functions are specified naturally using recursion. Understand the distinction between they uses the word recursion, as a recursive function in a

programming language, against the specification of a mathematical function using recursion.

So, and again observe that the word function means two different things at two different contexts, when I say mathematical function I mean a function as we studied in school, it is a function with the domain and a range etcetera, etcetera. So, do not get confused with or do not miss making the distinction between a function call, the definition of a function and the mathematical function.

So, these are all three different concepts and ensure the three is different concepts though they use the same words, do not overlap in your mind, this is extremely important. So, now, let us look at the factorial function, the factorial function is a mathematical function. Now, because it is a mathematical function, it is useful for me to tell you, what the domain of the factorial function is and what the range of the factorial functions is. The domain of the factorial function is the set of positive integers and the range of the factorial function is also the set of positive integers.

Therefore, let us look at the simplest values of factorial and let us even recall what factorial is, factorial of 0 is 1, factorial of 1 is 1, factorial of 2 is 2 multiplied by 1, which is 2 factorial of 3 is 3 multiplied by 2 multiplied by 1, which is 6. In general, factorial of n is n multiplied by the factorial of n minus 1, if n is greater than or equal to 1. If n is equal to 0, then factorial of 0 is given as a fact. It is the base case factorial of 0 is given as fact, it is not recursively specified, let us given as a fact and factorial of 0 is take it to be 1.

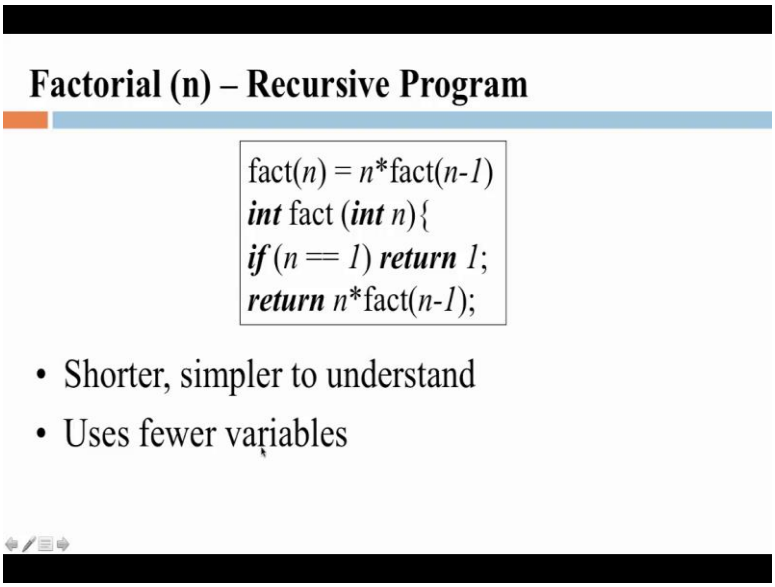
Now, such a definition of a mathematical function is called a recursive function, if you go back to your school mathematics, you will find that the greatest common divisor of two integers also has a recursive specification. Similarly, if you look at the binomial coefficients, they have recursive specifications and so on and so forth; there are many mathematical functions which are recursively specified.

Now, let us come to the world of programming languages and in particular programming language C. C supports programmers writing recursive functions, using recursive functions, mathematical functions which are recursively specified can very easily be converted into computer programs. I repeat this, the world of mathematics gives you many recursive functions.

C allows you to define recursive functions, that is, it allows you to define functions from the body of those functions, you can make a call to the function itself. So, these are recursive functions in a C programming language. Therefore, the mathematical functions which are recursively specified can be very easily converted into computer programs using the power of recursion provided by programming languages, in particular the power of recursion provided by the C programming language.

So, this is why recursion is extremely important, many things which are mathematically specify, many mathematical function which are recursively specified can immediately we converted into computer programs. So, keep your eyes open for mathematical primitives that you study either at school or college and see, which of them are recursively specified. And if you see something is recursively specified, immediately you can do this exercise, converting it into a computer program using the facility or the feature of defining recursive functions.

(Refer Slide Time: 10:16)



So, let us just look at a crisp recursive program for factorial. So, let us just recall one thing, the recursive program has to have well defined exit conditions, if you do not well define exist conditions, then a recursive program will run for ever, and exhaust your systems memory before it stops. Or rather it exhausts the amount of memory that you program has access to before the program will stop executing.

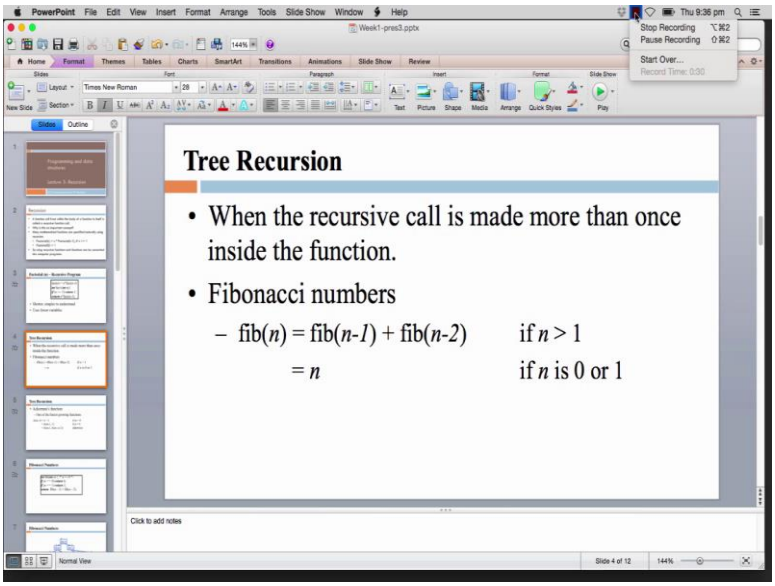
And remember that, this is not an infinite loop; I am not going into the details of what happens, where recursive program runs for ever. But, I just want to tell you that, when the recursive program, a poorly written recursive program that does not terminate is not an infinite loop, it is different from an infinite loop. An infinite loop essentially the CPU keeps running forever, when we little not terminate till certain bounty conditions or index conditions are met.

On the other hand, a recursive program which is poorly written will terminate, because it will run out of a certain kind of memory that your program has accessed while execution. So, what is the most important thing that I have said so for that, you have to have very well defined clear exit conditions in your recursive function definition. So, now, let us look at this recursive function definition, a parentheses missing this is small error here does not matter.

Let us just look at the definition of the function call fact, fact takes as an input argument or integer n, it returns an integer, as specified here, see if n is equal to 1, it returns the value 1. Otherwise, it returns the value n multiplied by fact of n minus 1, they must have been a parentheses here and that has been best. This basically encodes, the mathematical definition of the factor will function same the fact of n is n multiplied by fact of n minus 1.

One must understand how a recursive program executes and that is the focus of today's lecture. However, as you can see once for understands how recursive functions are evaluated at run time a recursive program is often simpler to understand, but it is very important you must get use to the concept of how recursive functions are evaluated at run time. And typically, recursive programs also usese smaller number of variables, though this is not something that we will dwell upon.

(Refer Slide Time: 13:33)

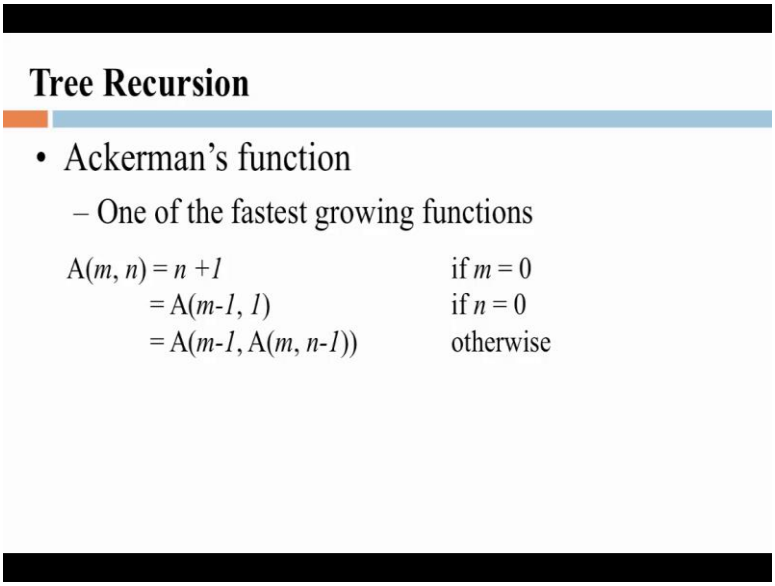


There is another kind of recursion, which we call tree recursion, it does not matter, why it is called tree recursion, it is a concept that you will become clear with, as we go through this course. When a recursive call is made more than once inside the function, then one potentially gets a special kind of recursion that is best represented using a non-linear data structure. I repeat this, certain mathematical functions are specified recursively, using the values of the function on more than one smaller value. I repeat this, certain mathematical functions are specified recursively, using the values of the function on more than one smaller value.

As opposed to factorial, factorial just depends upon n the factorial of n minus 1, factorial of n depends upon n multiplied by factorial of n minus 1. So, factorial of n really depends upon factorial of n minus 1. On the other hand, let us look at the Fibonacci numbers, which as specified by the following recurrence equation, this the mathematical function, sure all of your familiar with the Fibonacci sequence, the numbers in a sequence are call Fibonacci numbers or Fibonacci numbers and if n is 0 or 1.

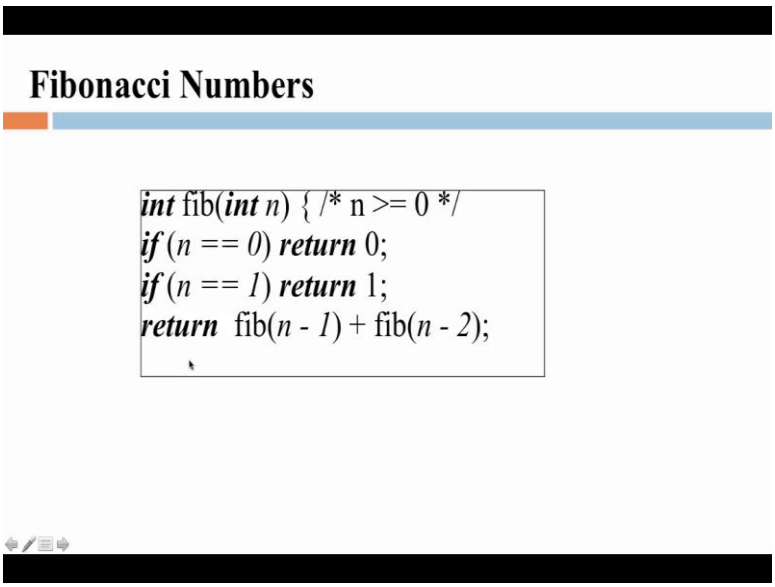
So, this is the example of tree recursion, where a recursive call is made more than once. So, in the mathematical definition of the Fibonacci numbers, the nth Fibonacci number is the sum the previous two Fibonacci numbers immediately previous two Fibonacci numbers and the base values are the Fibonacci numbers for n is equal to 0 and 1 or 0 and 1 respectively.

(Refer Slide Time: 15:59)



So, these are examples of tree recursion and similarly here is the Ackerman's function which is another recursively specified function, its one of the fastest growing functions. I will encourage you to actually plot the values of these functions using an excel spread sheet. You can actually do this manually for different values of m and n to get an understanding of how fast the Ackerman's function actually grows.

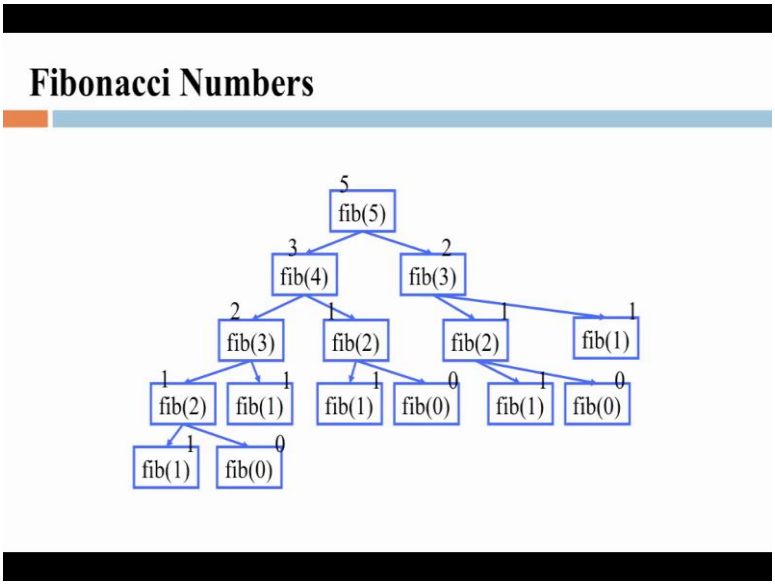
(Refer Slide Time: 16:27)



So, this is a small repeat of the Fibonacci function specification in the C programming language again the flower brackets are missing, it says, observed that there are two if's

right which very clearly specify the return conditions, if the argument n is equal to 0, then you return a value 0 and if it is equal to 1, then you return a value 1. Otherwise, you return the sum of the Fibonacci number n minus 1, plus Fibonacci number n minus 2 which is now recursively calculate it. Then, just after this the parentheses should have been there.

(Refer Slide Time: 17:13)

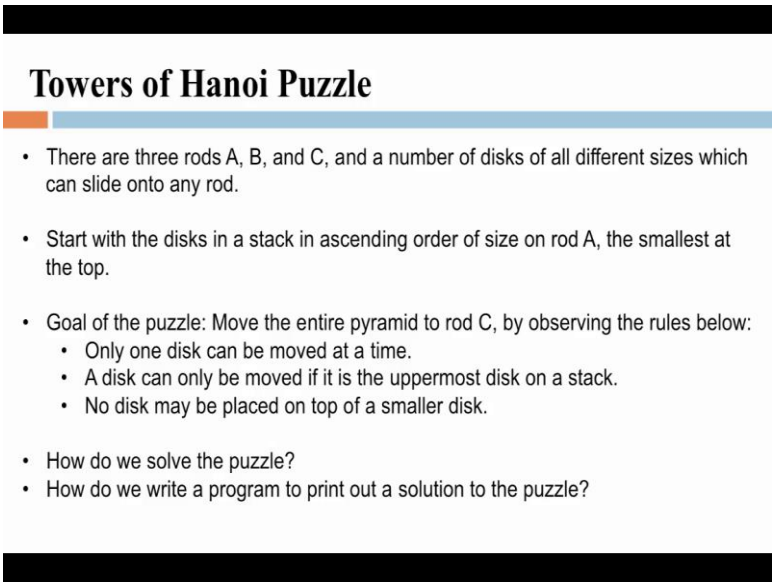


Let us visualize this as a tree like structure, and let us look at the 5th Fibonacci number, the 5th Fibonacci number is the sum of the previous 2 Fibonacci numbers, it is a 4th Fibonacci number, it the 3rd Fibonacci number which in turn is a sum of the 3rd Fibonacci number in the 2nd Fibonacci number. And the 3rd Fibonacci number is sum of the 2nd Fibonacci number, in the first Fibonacci number and the 3rd Fibonacci number again here is the sum of the 2nd in 1st, the 2nd Fibonacci number is the sum of the 1st Fibonacci number are the 0's Fibonacci number and so on.

So, as you can see there is this very nice full binary tree structure. So, by full binary tree I mean that, for every node there are exactly two children, except for the leaf which have 0 children. So, for example, this node has exactly two children and this node has no children at all, it is a leaf node and observe that at these leaf nodes you get the base value, that is the 0th Fibonacci number has a value 0, the first Fibonacci number has the value 1. And these two values are added and this is the value of the parent Fibonacci number, which is the second Fibonacci number and so on.

You can see that the 5th Fibonacci number has the value 5. So, therefore observe that the evaluation of the Fibonacci number recursively is, same as evaluating a tree like structure, where the leaf nodes has some values and these values are propagated up words towards a root of this tree. And this is an example of a function, which is evaluated by visualizing it as a tree and this is what I mean by saying that certain mathematical functions are viewed in the world of tree recursion.

(Refer Slide Time: 19:29)



Now, let us come to the programming exercise of the day and it is a very famous puzzle and there are two things here. In the rest of the lecture I am going to try and show you, how to think recursively and how to also program recursively or how to program recursive functions. So, let us just study the towers of Hanoi puzzle for a short time.

So, in this puzzle there are three vertical rods I call these rods A, B and C and there are a certain number of disks, each of these disks has a different radius from the other disks and each of these disks hand slide into any of the rods. Now, the exercise in the following initially all the disks are arrange in a stack in ascending order of size in the rod A or the tower A. This is the first tower and all the disks are arranged in a stack like fashion, one on top of the other with the smallest disk at the top and the largest disk at the bottom of the tower.

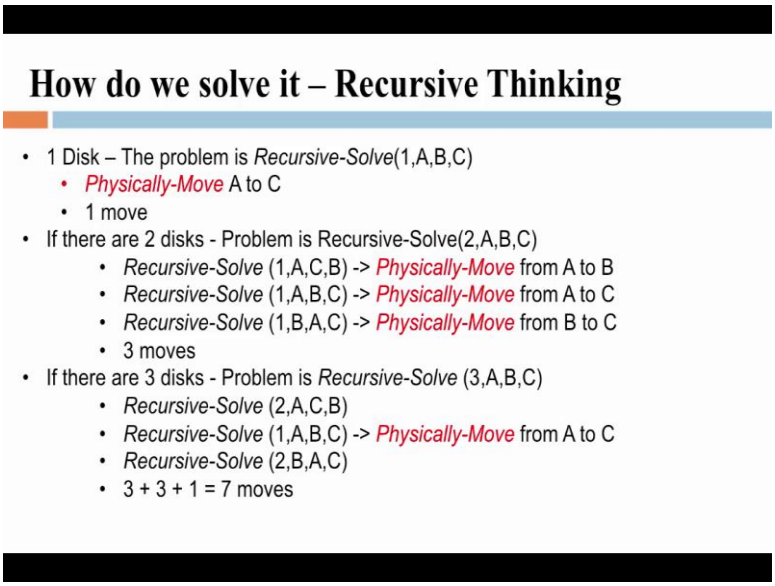
So, in other words all these disks are arranged in a pyramid like structure, one tower or one rod and this rod let us call it A. The goal of the puzzle is to move the whole

parameter to rods C and one has to observe three rules which are listed below and any step only one disk can be moved and only the disk at the top of a tower can be moved, that is only the smallest disk can always be moved, that is a smallest disk in any of the three towers can be moved and at no point in this movement can a disk be placed on top of a smaller disk.

So, therefore, the condition that is placed on the movement is that at any point of time if you stop and look at the snap shot of the three towers, what you see are pyramids whose sizes are decreasing from the bottom to the top. The disk sizes are decreasing from the bottom to the top and this property must be satisfied at every point with the movement of the disks from tower A to tower C, so this is a puzzle.

So, any of you can pick up I mean the towers are not, so important you can get three disks or four disks are certain number of disks all of different radii and place them on the floor and mark the towers using some chalk piece on a floor and you can play this game. So, what is the solution to this puzzle and secondly can you write a program that can print out to a solution to the puzzle. Therefore, there is a two exercises here how does will solve the puzzle is one exercise and the second exercise how do you write a program to print out a solution to the puzzle. So, let us deal with a first problem here, how does one solve the puzzle, I am sure each of us first will attend multiple things, this being a short lecture I will actually directly try to in inculcate a certain thought process in your mind.

(Refer Slide Time: 23:15)



And I am trying to put in the idea of thinking of recursive sub problems, so that is the whole idea. So, I am going to try to suggested that you take the problem and see if you can solve the problem if you know how to solve, smaller sized problems that is let us assume that you want to solve the case of the problem, when you have 3 disks in tower A. Now, if you know how to move 2 disks, then can you solve the case of the problem when you have 3 disks, so this is the idea of recursive thinking.

So, I repeat this again you want solve a problem of a certain size, you ask can I solve the problem, if I know how to solve instances of the problem of smaller size. So, let us start off with a simplest case, let us assume that we have only one disk, the disk is in tower A and you want to move it in to tower C, it is very simple. So, I am going to motivate you to think about a function called recursive solve, the first argument for recursive solve is a number of disks which are in the first tower, the tower is A, B is the intermediate tower and C is the destination tower I repeat.

So, in this recursive thinking thought process, I am going to tell you to think about a recursive function called recursive solve. The first argument to this function is the total number of disks that you have. The second argument is the tower that contains these disks, that is in this case the tower A, C is the destination disk that is, you want to move all these disks into tower C from tower A and you can use tower B as the intermediate tower.

Now, here when you have only one disk the problem is very easy, you do not use the tower B at all simply physically move A to C. Physically move is again a function, which is always applied in the case when you have to move only one disk. Now, if there are 2 disks let us think of the problem recursively, now you know how to solve the problem then you have exactly 1 disk to move.

Now, we have to solve the problem when you have 2 disks in tower A, you want to move it into tower C and you are free to use intermediate tower B. How does one think of it? The way to think of it is to close your eyes and imagine that the biggest disk does not exist at all. So, now what do you do? You recursively solve assuming that you have moving only 1 disk, which disk the top most disk in tower A to tower B via tower C.

And we know that, if you need to move only one disk it is the base case, you simply move from the source to the destination, that is what we did here. Whenever you have one disk you move the disk from the source to the destination directly. Similarly here, so, that is only one disk you want to move it from A to B via C individually physically move it from A to B via C. Now, the next step would be to move the disk in tower A, there is now only a single disk after this step, that is the largest disk.

Now, you move that largest disk from A to C via B, because that is the single disk is a physical move directly from A to C and the last step would be to remove the second disk, which is a smallest disk, which is now in tower B to C again directly from B to C. Therefore, we have performed three moves observe very importantly you want it to solve the case when you have 2 disks and observe that you have solved the problem three times, when there is only a single disk.

So, therefore, you recursively solved the movement problem for 2 disks using three function calls to the movement problem, when there is only a single disk. So, now, let us try and see how to do this when you have 3 disks. So, what does recursive solve look like when you have to move 3 disks from the source tower A to the designation tower C via B. The idea is very simple, you recursively solve the problem assuming that there are only 2 disks, that is ignore the fact the third disk as a the bottom of tower A, it is a biggest disk ignore it.

Imagine that you have moving only 2 disks, the top 2 disks from A to B via c. So, this is the black box recursive call do not worry about how it works, you know how to do it

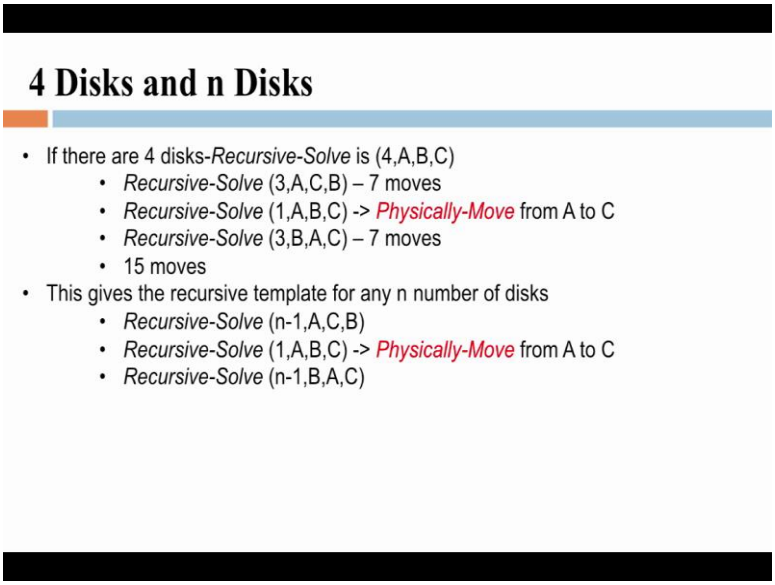
from here. So, now, you recursively solve the case when you have 2 disks for move from A move it to B via C. So, therefore, A is now considered as a source tower B is a designation tower and C is the intermediate tower, observe that this is the recursive stack.

So, this is one of the step that you perform to solve the problem when you have 3 disks to move, where the source is A, C is a designation and B is the intermediate thing to solve this, first you say I am going to think of A as a source, B is the destination and C as the intermediate tower and I am going to move only the top 2 blocks which are in tower A or the top 2 disks which are in tower A. Let us assume this has been successfully done.

Now, what do you see in tower A that is only one block, you move it all the way to C. So, that is the physical move this is the base case move A to C directly, now you have a two blocks in B, you want to move then C and you move then via A. So, your source now becomes B, your destination becomes C which is the final destination in any case, then you move it via A and you solve this sub problem of by considering in the case where there are only two blocks.

As you can see recursive solve on 2 blocks states 3 moves that is what we have seen here, recursive solve when you have only one block or one disk states only one move and then a recursive solve on 2 blocks or 2 disks takes another 3 moves a total of 7 moves. So, that is 2 power 3 minus 1.

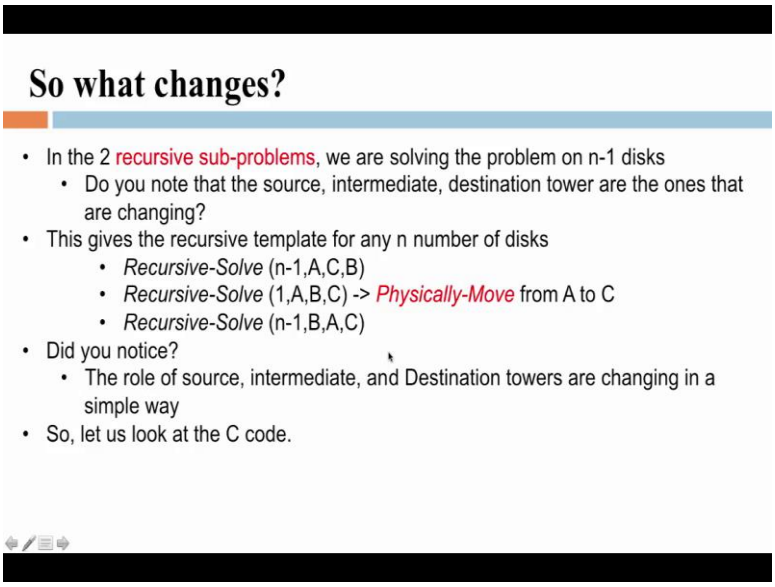
(Refer Slide Time: 30:26)



So, which is generalizing is to 4 moves and 4 disks and n moves. So, if there are 4 disks recursive solves 4 disks from A to C via B and I have this is clear now. First step do recursive solve for the top 3 disks in A from A to B, the intermediate block intermediate tower via C, this will takes several moves that is what we studied in the previous slide, then you move the bottom most this which is now in tower A, it is completely exposed. Because, everything is being move to tower B and you move it to c that is a physical move and that is described does in recursive function call.

Of course, B does not get used here and then you now make B as your source. So, reminding 3 disks are now in B, you move them to C via A which is now empty and that takes another several moves and I am sure you can very easily generalize this to the case, when you want to move n disks from A to C via B.

(Refer Slide Time: 31:47)



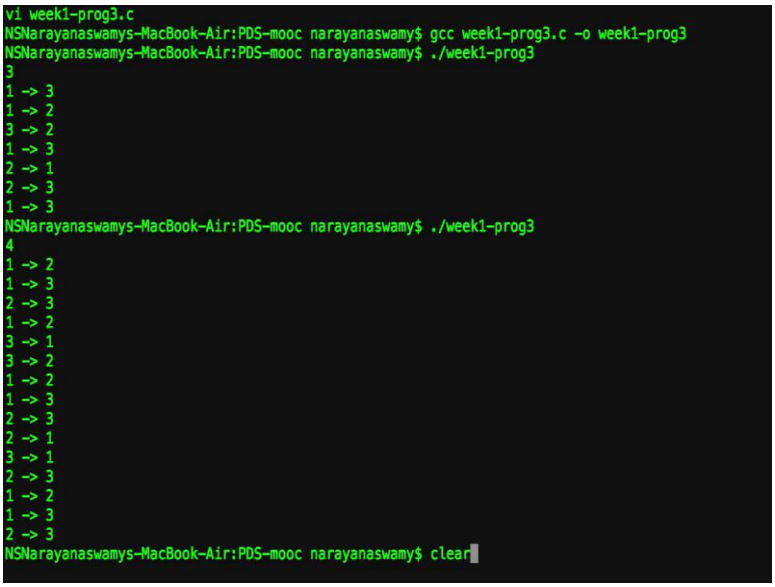
So, let us just understand the whole thing again, it is extremely important for you to understand that you have not yet found the sequence of moves, you only recursively said that the sequence of steps that are required to move n disks from tower A to tower C can be obtain by recursively finding the sequence of steps to move n minus 1 disk from A to B via C, then by performing that is this single physical move of the larger disk from A to C and then recursively moving the disks from B the n minus 1 disks from B to C via A.

Observe that to be able to find the sequence of moves for every n, you have to write down this recursion tree that we spoke about, this is also going to be a binary tree and if

you list down the leaf nodes in an appropriate order. You will essentially get the whole sequence of moves that need to be perform to achieve our goal are moving n disks from the tower A to tower C, this is what our program is going to do and what is the most important thing.

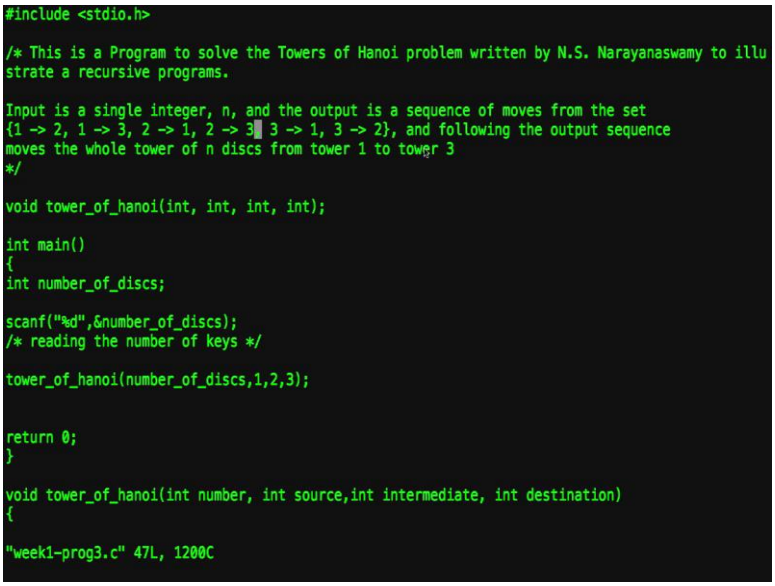
So, the role of source, intermediate and destination towers are repeatedly changed in a certain simple manner in the description of our recursive algorithm or the recursive function. So, now, let us go to the C program and let us look at one execution and this will finish today's lecture.

(Refer Slide Time: 33:59)



As you can see I have already try running this program a couple of times.

(Refer Slide Time: 34:15)



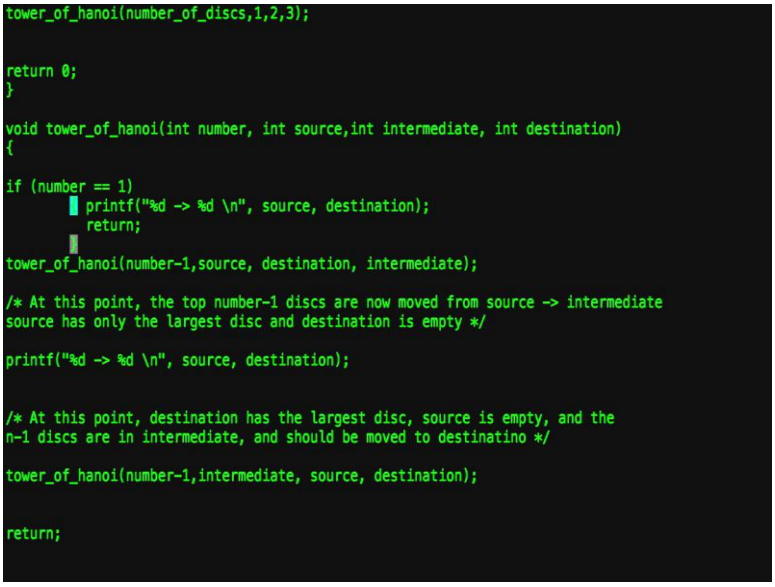
So, for in the previous lecture you will notice that I have removed stdlib dot h and string dot h it is unnecessary for us to include these preprocessor directives, all that I need is simple printf and scanf functions, which are available in this header file stdio dot h. So, this is a very simple program to solve the towers of Hanoi problem, the input is a single integer n and the output is a sequence of moves.

Now, what is the sequence of moves, every move is essentially one of the 6 possibilities, this one says move from tower 1 to tower 2, which one we will move only whatever is at a top of the tower and where do you move it in tower 2 to the top of tower, then this one says move from tower 1 to tower 3, tower 2 to tower 1, tower 2 to tower 3, 3 to 1 and 3 to 2. Therefore, one has to output a sequence of moves, each move is from this sect and each moves specifies from which tower to which tower you move a disk.

And if you follow the output, this output sequence one after the other, that is output by this program one moves the whole tower of n disks from tower 1 to tower 3 satisfying all the conditions that are outlined in a towers of Hanoi problem. This is the function prototype, the function prototype has 4 integer arguments, the first integer argument as you will see is for the number of disks which are there, the second integer tells you the tower number the, so does a third one and the fourth integer argument tells you which are the tower numbers.

And in this program we assume with the towers are number 1 to 3 1 to and 3 are the tower numbers. So, here is the main function number of disks is the integer variable, we read the number of disks that we want to solve the problem within this is the number of disks are not the number of keys in previous program and here is the function call. So, the function was call tower of Hanoi, the first argument is a total number of disks which you want to move from h 1 which is the source tower to the designation tower 3 via the intermediate tower 2 finally, you return 0 on successful completion.

(Refer Slide Time: 36:54)



So, now let us look at the recursive function, it form as a same template of what we discuss, when we looked at the slides except that we have strategically put in appropriate print statements. So, let us just look at the sequence, if the number of disks is 1 what do we do, we performed a moves, where do we move from, we move from source to destination directly. Therefore, that is what this print says, then when makes recursive call if the number is more than one.

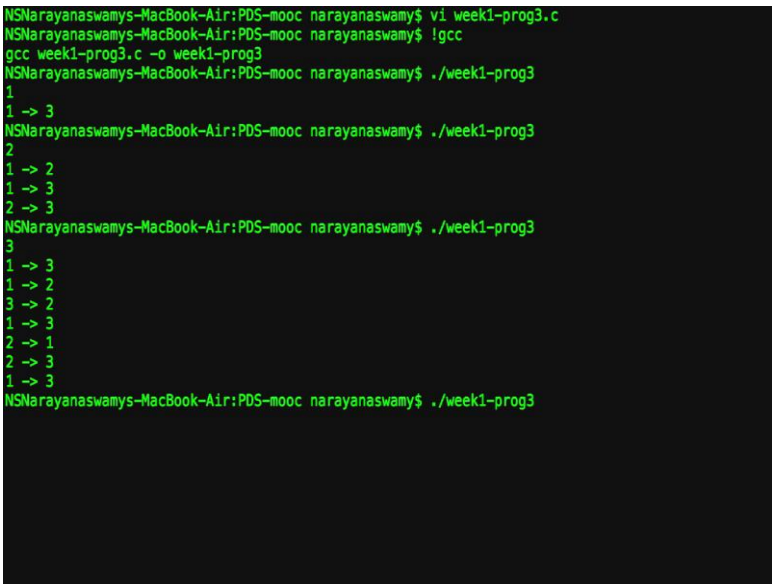
So, you move from source to destination then you return to the place from via a function call was main, if number is equal to 1. So, the control will reach this statement if the number is actually more than one, if number is more than one then there are at least two blocks. So, you solve the tower of Hanoi problem on two blocks moving them from source to intermediate that is you move the two blocks from source to intermediate via destination. Observe that this is the recursive function call.

Now, once the execution of this function is completed and the control has returned to this point. Let us look at the property that is satisfy, the top n minus 1 disks have now will move from source to intermediate and the source has only the largest disk and the destination is empty, this is the property after executing this function. Now, the source has only one disk, so we move this directly from source to destination at this point the destination has the larger disk, the source is empty and the n minus 1 disks are in the intermediate tower and they should be move to the destination tower.

How is this achieve? This is achieve by making a recursive function called the same function, same then you want to move those these many number minus 1 number of disks from the intermediate tower to the destination tower, where are the source tower. And once this is done at the top most level of recursion, you can return and all the towers would had been move from source to destination via intermediate and this quite here that, because the each of the moves is always moving a disk from the top of one tower to the top of other tower and it is easy to observe that the conditions are satisfy.

So, the best way to observe this is writing of formal proof which we will do if you were doing an algorithms course, this you actually test this program. So, let us just compile and run this program.

(Refer Slide Time: 40:12)



So, this expects are integer and let us assume that I get only one disk, then it is says move 1 to 3 directly this is fine. Now, let say there are two disks and before I get return

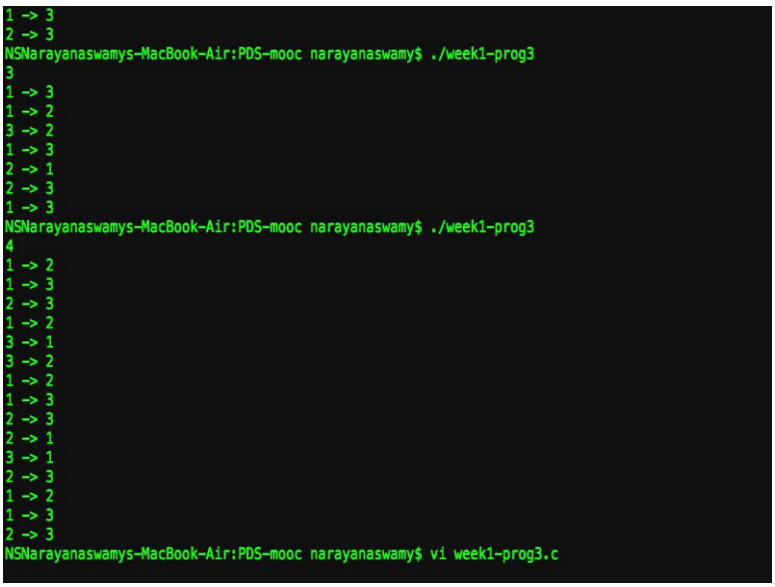
let us just verbalize the plan and let us just state the plan. So, the idea would be that there are 2 disks in tower 1 in they all need to be move to tower 3.

So, the idea would be to move the top most disk to the 2nd tower then move the bottom most disk to the 3rd tower and then move the single disk, which is the smallest one which is now in the 2nd tower to the 3rd tower, that is the plan would be 1 to 2, 1 to 3 and then 2 to 3 let us check if this is right, 1 to 2, 1 to 3 and 2 to 3 and encourage you to use a note book at test check whether this is right.

So, let us run the program for 3 and then stop which say let us just take a look at this, you want to move 3 disks from the 1st tower to the 3rd tower. So, what we do is you first take the 1st disk and move it to the 3rd tower, this is the top most disk, then we move the 2nd disk in tower 1 to tower 2, then you move the top most disk which is now in tower 3, the smallest disk which is in tower 3 to tower 2 at this point of time observe that the top 2 disks which were there in tower 1 have now move to 2 which is the intermediate tower.

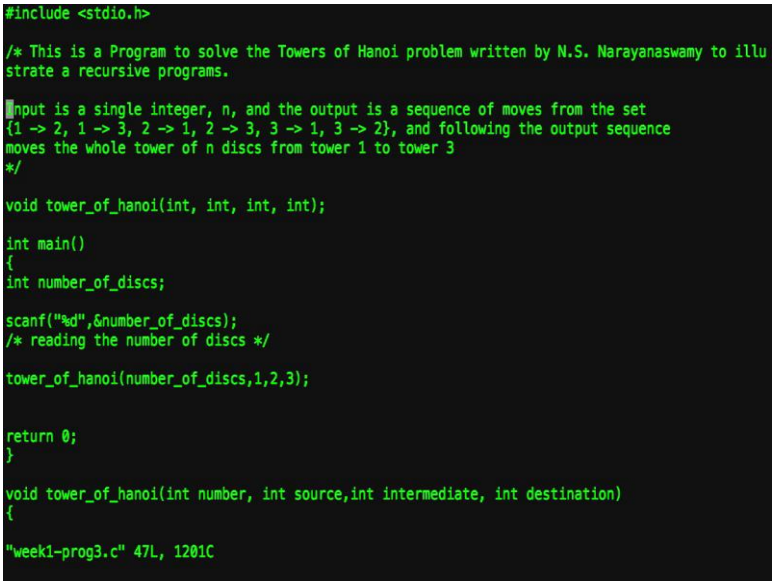
Now, you move the largest disk from tower 1 to tower 3 and now you move the 2 disks from tower 2 to tower 3 via 1 does an intermediate tower. So, you move the smallest disk from tower 2 to tower 1, then the second smallest disk directly goes from tower 2 to tower 3 and then the smallest disk goes from tower 1 to tower 3. So, as you can see this program does what for the case when the number of disks is 3, the same you can check for 4 and 5 onwards.

(Refer Slide Time: 40:51)



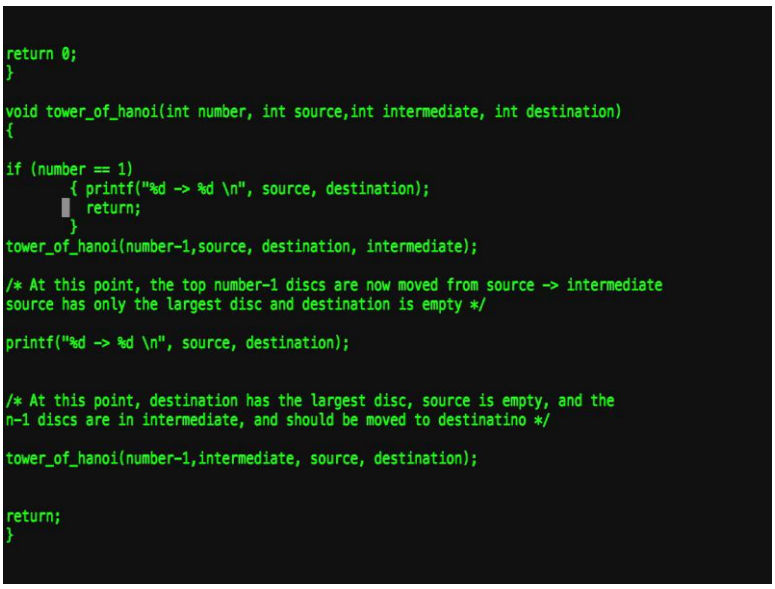
So, we are not doing a formal proof, because this is not the focus of the course.

(Refer Slide Time: 43:11)



The important concept that is illustrated to you here is that a fairly sophisticated plan, indeed a plan which has almost 2 power n, where n is a total number of disks actually there are 2 power n minus 1 moves is very succinctly in very crisp fashion is specified using a very small recursive function.

(Refer Slide Time: 43:30)

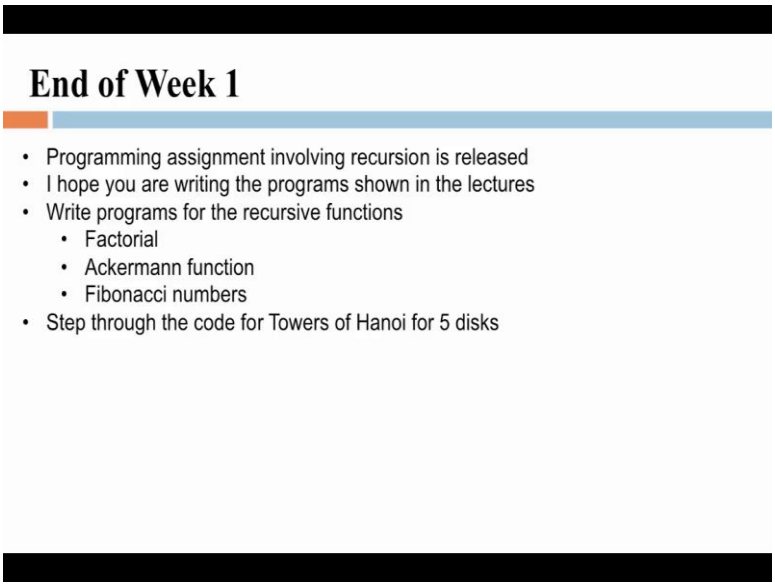


So, observe that this is the whole recursive function. What is this recursive function encode? The recursive function basically encodes the fact that to solve the problem on n

disks, somehow move the top n minus 1 disks into the intermediate tower or rather recursively move the top n minus 1 disks into the intermediate tower by using the designation tower as the intermediate tower. And then move the largest disk which is now available at the top of the source tower to the 3rd tower which is our destination tower and then move from the intermediate tower which is our 2nd tower to the 3rd tower again recursively via the tower 1.

So, this is exactly specified these are the recursive function calls to move from source to the 2nd tower via the destination tower. And the third one says move whatever has been accumulated in the intermediate tower to the destination tower via the source tower. And these are the well defined exit criteria from the recursion; otherwise, a recursion will not terminate. So, this more one is finishes the presentation for today.

(Refer Slide Time: 45:00)



Let us just go back to our slides and observe that we have come to the end of week 1 and it has actually being a very hit tic week we have learned or review very important concepts in this week. And therefore, each of you should have a steep learning curve as you go through this week, this is probably the toughest week and once you understand how to actually implement these in the C programming language.

The remaining weeks are going to be the definition of different data structures and then setting up appropriate programs to use them to solve different kinds of problems, every data structure is useful for a certain kind of a problem. Therefore, understanding the data

structure is very important. So, that you can implement the programs for the appropriate problem and being able to program in C is the way by which you will really write those programs and feel comfortable an accomplished having learn these data structures.

Therefore, this week is very likely to be or toughest week where the many new concepts which you probably have only studied in text books I am encouraging you to write programs for all of them. So, therefore, you will notice that the first programming assignment involve in recursion, involves binomial coefficients that has been released. And I hope that you are writing on the programs that I have shown in the lectures at least those if not more.

And today's lecture you have seen three recursive functions for which have not shown you programs. I encourage to write recursive functions in C for these three particular recursively specified functions and also run through the code for the towers of Hanoi for 5 disks. So, for this I encourage you to write the towers of Hanoi program in C completely by yourself, exactly the recursive function do not Google the code or do not the search the internet for the code, go through the video trying understand the exactly what I have said the recursive thinking part of it and try to write the C program with the appropriate exit conditions, that is the best way to learn.

Of course, you can choose to cut and paste from an appropriate web page of the internet and compile and run, but that is not the best way of learning. So, thank you very much that brings us to the end of first week, it has been an even full week for all of us and it is also a time when we are understanding all the challenges with uploading the videos and getting the assignments ready on time and so on and so forth. Hopefully the subsequent weeks will be much smoother and you will find the course even more enjoyable.

I will stop with the very important piece of advice, there is no better way to learn programming than to write programs. So, please do not watch this video as you watch a movie or a TV program and go back and read a text book. So, that is a worst way to learn the subject you will not learn anything, the best way to learn this subject is to get your computer, sit down put your fingers on the keyboard have a note book by your side, understand the logic and write programs. After the first time you write a program it by not even compile there will be errors, you have to file through it and make a programs worth.

Thank you very much for listening to the small piece of advice, hope you enjoy this week bye.