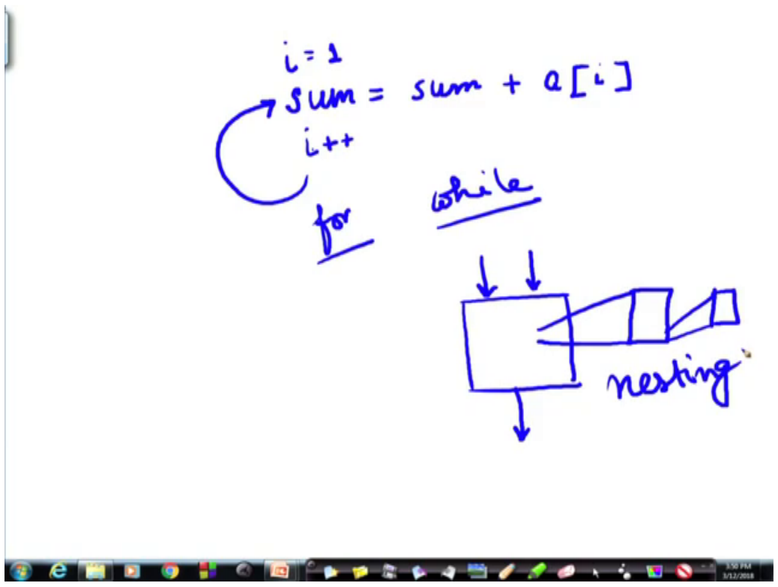
Problem Solving through Programming In C

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

Lecture – 53
Recursion

Today, we will be discussing on a new concept of programming which is very interesting, but possibly not very much familiar to you. That is known as Recursion. You know repetitions; how they are implemented in C programming. For example, whenever I want to do a particular work.

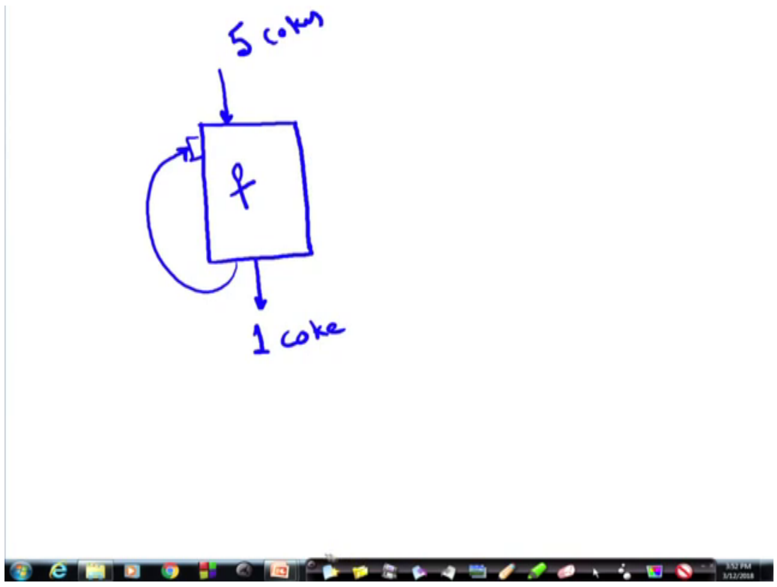
(Refer Slide Time: 00:55)



For example, I want to add 10 numbers, then I add the sum with sum plus sum a i with i equal to 1.And then I repeat it i plus plus and I go on repeating it and this repetition is done in the form of a for loop or while loop you know that right.

Now, recursion is a different way of doing this. When a function calls itself; that means, a function; a particular function will have some inputs and we will deliver one output. Now a function has been now nesting of functions that in order to achieve this objective of taking these inputs and delivering these output, a function could call other functions from their another function and then ultimately, return to this function return back here in that way, it could be done this was known as nesting.

(Refer Slide Time: 02:24)

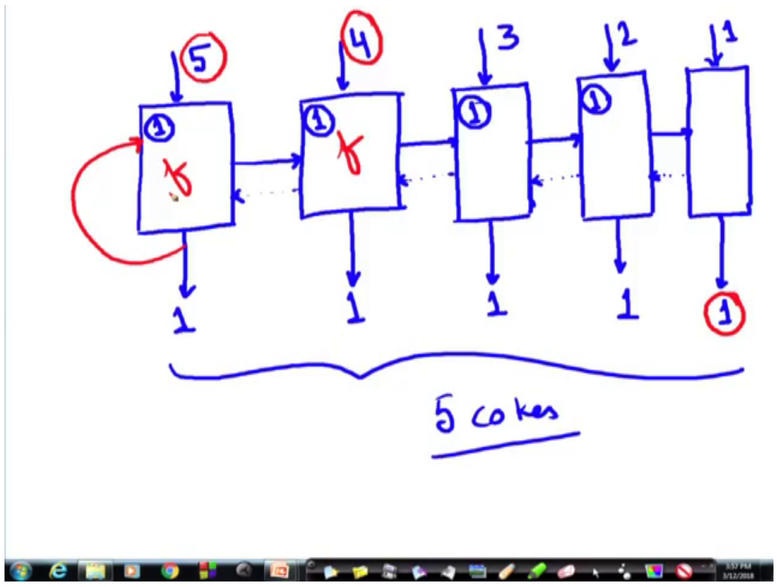


But recursion is a little different, it is that here there is a function which is supposed to be to deliver some output. Now in order to do this, it is actually repeating itself; that means, let me give you an example. Suppose, I have a machine which can generate 1 apple, all right, it can generate 1 apple, produce 1 apple or maybe say apple let us say, it produces a vending machine which can deliver 1 coke, alright, 1 piece of coke, 1 can of coke, right.

So, you put in some input some commands and it gives you 1 coke. Now, it has been asked to deliver 5 cokes; 5 cokes. So, how can you deliver 5 cokes? It can deliver only 1 coke at a time. So, as we know in our knowledge of iteration. So, this is some function; function is delivering coke. So, it can be repeated, this switch can be pressed, there is a switch, it can be pressed 5 times. So, first, 1 coke comes out, second coke comes out so; that means, this is being repeated 5 times that is the conventional 1 coke.

So, that is the conventional way of iterating repeating the same thing in order to get the it 5 cokes let us try to do it in another way, let us assume that this machine can deliver 1 coke at a time, but it can also clone itself. So, let us see how it looks like.

(Refer Slide Time: 04:33)



So, I need 5 cokes. So, 5, but I can deliver only 1. So, I saw that ok, I can deliver only 1 coke, all right, I can I will deliver 1 coke, but I will not deliver it till I am ensured that all the 5 cokes are deliverable.

So, what it does? It clones itself makes another one copy of itself say and activates this with I can deliver one coke. So, I asked him to give a 4 cokes. Now this one can also deliver only 1 coke and finds that ok, I cannot deliver everything. So, I keep 1 coke ready, what I can, but I cloned myself and another copy, I activate him this machine and say please deliver 3 cokes, but it cannot also deliver 3 cokes, it can deliver only 1. So, nothing is being delivered, but only kept noted. So, it puts somebody another clone please deliver 2 cokes, I have got 1.

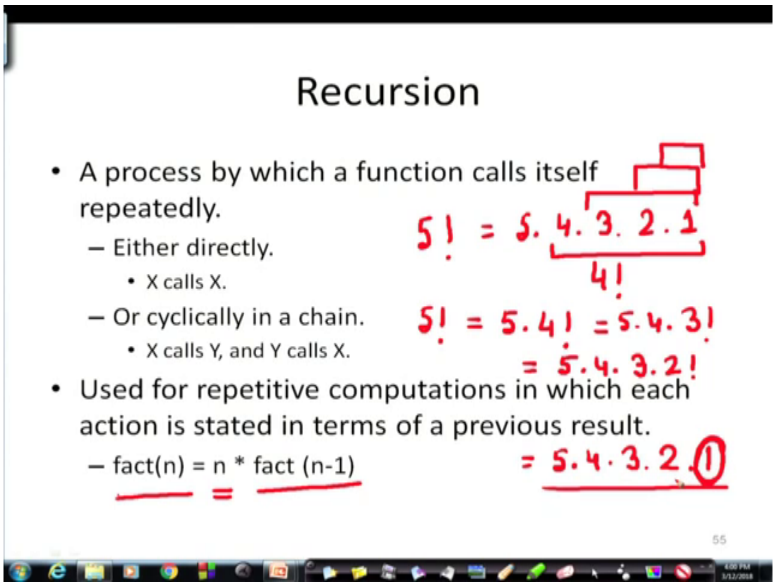
So, it is also not delivering because it is not being able to satisfy the request of 2 cokes which has opposed to it by this machine. So, it now comes to another clone. So, asks this clone please deliver 1 coke. Now as you know this machine can deliver 1 coke. So, now, it delivers 1 coke and tells ok, I have delivered. Now this one, then since it has delivered, it delivers the other coke which it could do. It also tells that its requestor it is earlier pro, I mean after the earlier version that I have done my thing you do yours. So, what it will do it can only each can remember each can only deliver 1 coke at a time.

So, it will deliver another coke, it passes on to the again tells its caller or its generator that deliver another coke deli I have delivered. So, it delivers the other coke and ultimately this one also knows that its child has delivered the coke. So, ultimately we get 5 cokes. So, you can very easily see that if I had this was one this is one way in which if each of this function if each of these blocks is the same function, but it is being activated with different requests and they are waiting till the request can be fulfilled.

But in the meanwhile passing on the request to another one and as soon as this could fulfill it passes it on and then it goes back in the case of iteration what would have happened this would not be done thus this same thing 1 coke and again call this and if another coke another coke like that. So, if I call each of these as functions, then actually this function. And this function, there is no difference between them only difference is a value with which it is being called and this process is successful because ultimately there will be a situation when this function of this machine will be able to deliver what is has been asked to 1 coke it can deliver. So, that will be done.

Therefore, we can go back and have everybody else deliver the same thing this is the principle of recursion we will I just used it as a fun example, but let us now come to a little more serious look at this.

(Refer Slide Time: 09:26)



Recursion

- A process by which a function calls itself repeatedly.
  - Either directly.
    - X calls X.
  - Or cyclically in a chain.
    - X calls Y, and Y calls X.
- Used for repetitive computations in which each action is stated in terms of a previous result.
  - fact(n) = n * fact (n-1)

$5! = 5.\underbrace{4.3.2.1}_{4!}$

$5! = 5.4! = 5.4.3!$

$= 5.4.3.2!$

$= 5.4.3.2\,\textcircled{1}$

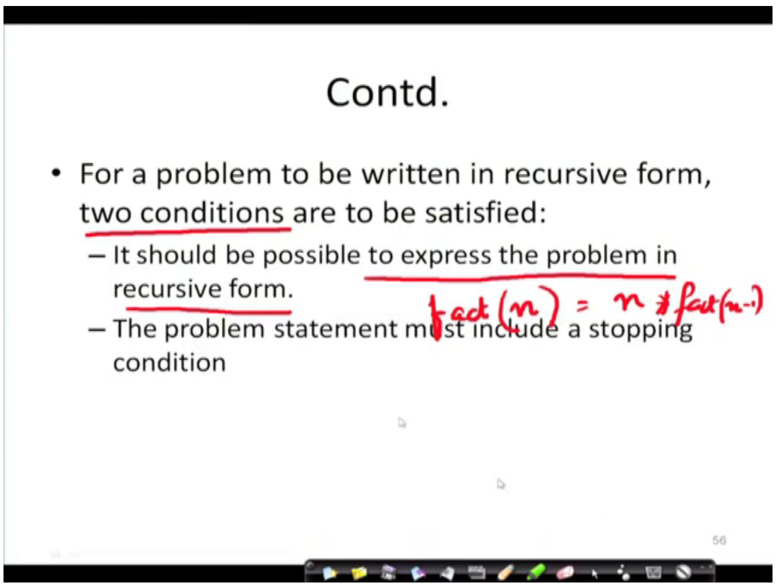
55

So, it is a process by which a function calls itself repeatedly, either directly like X is calling X or cyclically in a chain X is calling X calling Y. Y is calling X like that right used for repetitive computations. So, the best thing is you look at this example factorial in all of us know that factorial 5 is nothing, but 5, 4 times 4, times 3, times 2, times 1.

Now, you see the same thing, I can say; what is this part 4 multiplied by 3 multiplied by 2 multiplied by 1; this factorial 4. So, you see I am expressing factorial 5 factorial function in terms of itself 5 times factorial 4 and factorial 4 can again be expressed as 4 times factorial 3 and factorial 3 can be expressed as 3 times factorial 2 and factorial 2 and be expressed in terms of factorial 1. So, 5, 4, 3, 2 and factorial 1 is the end. So, that is 1. So, I am certainly getting the factorial there without any further expansion.

So, this is how. So, we can write in general factorial of n is n times factorial of n minus 1 and factorial of n minus 1 will be n minus 1 into factorial of n minus 2 factorial of n minus 2 will be n minus 2 into factorial of n minus 3 in that way it will go on, but it will be successful only if there is a terminating point where which we often call the basis condition ok, I hope you have understood this.

(Refer Slide Time: 12:00)



Contd.

- For a problem to be written in recursive form, two conditions are to be satisfied:
  - It should be possible to express the problem in recursive form.
  - The problem statement must include a stopping condition

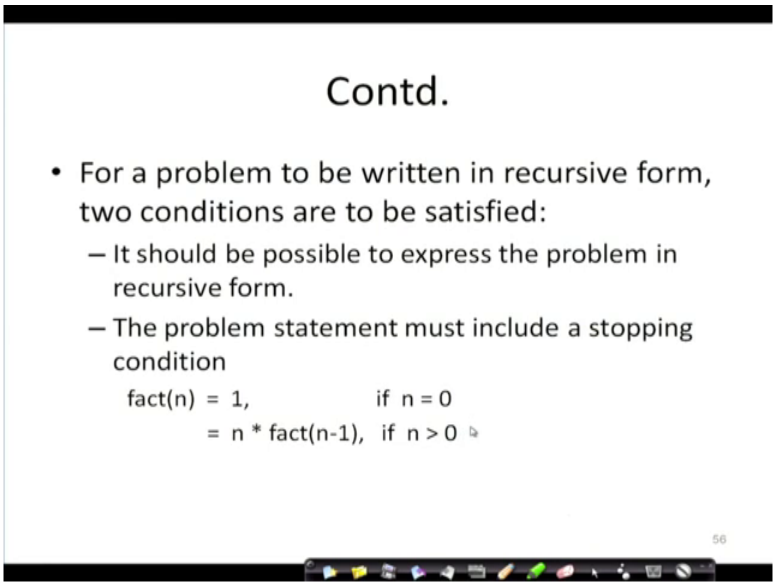
$fact(n) = n * fact(n-1)$

56

So, two conditions are to be satisfied in order that we can write a recursive formula. One is it should be possible to express the problem in the recursive form just like factorial n is n times factorial n minus 1 and also there should days another point that is the problem statement must include a stopping condition, what was my stopping condition in the case of factorial factorial 1 is 1 ok.

So, that was the stopping condition; what was the stopping condition in the terms in terms of in the example of delivering the coke when the machine was asked to deliver only 1 coke, then it can complete. So, that is a stopping condition and then we go back go back, all right.

(Refer Slide Time: 13:10)



Contd.

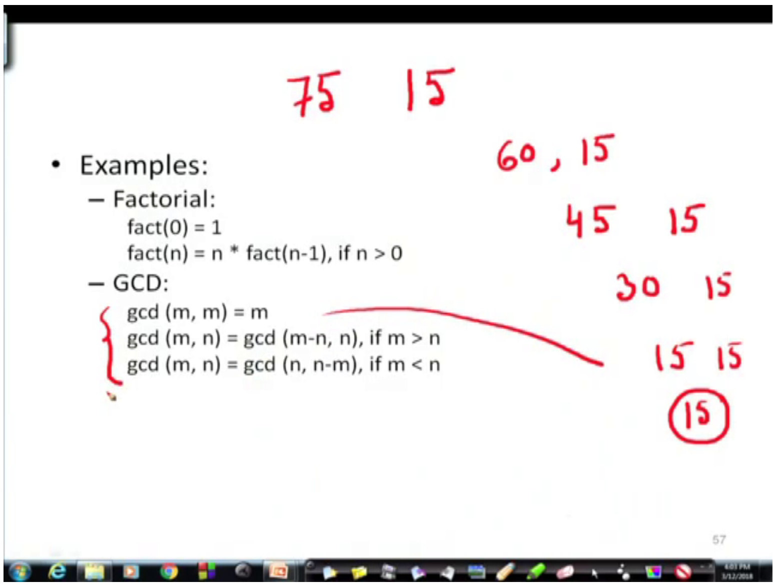
- For a problem to be written in recursive form, two conditions are to be satisfied:
  - It should be possible to express the problem in recursive form.
  - The problem statement must include a stopping condition

```
fact(n) = 1,               if n = 0
        = n * fact(n-1),   if n > 0
```

56

So, the stopping factorial n is 1 if n is equal to 0 or n equal to 1, otherwise, it is n into factorial n minus 1 if n is greater than 0. So, ultimately, it will go on and ultimately, it will conclude; example factorial 1 we have seen that.

(Refer Slide Time: 13:42)



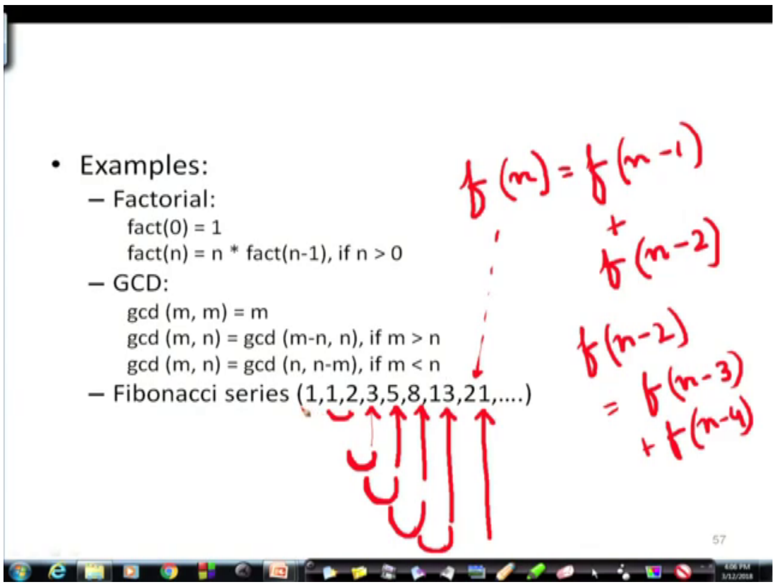
- Examples:
  - Factorial:
    fact(0) = 1
    fact(n) = n * fact(n-1), if n > 0
  - GCD:
    gcd (m, m) = m
    gcd (m, n) = gcd (m-n, n), if m > n
    gcd (m, n) = gcd (n, n-m), if m < n

75 15
60 , 15
45 15
30 15
15 15
15
57

Another example; greatest common divisor, we can express that in a recursive form, it is very interesting, you can look at it that greatest common divisor of the same number is itself that is the key to the logic GCD of m and m is m and GCD of m and n is GCD of m minus n and n or the other way. So, for example, GCD of 15 and 5 also 75 and 15 75 and 15 will be GCD of 60 and 15, GCD of 60 and 15 would be for GCD of 45 and 15, GCD of 45, 15 would be GCD of 30 and 15 GCD of 30 and 15 will be GCD of 15 and 15. Now I have got the stopping condition that GCD of the same number will be 15.

So, my result will be 15 is it clear? So, that is a recursive definition of GCD. So, most of the interesting problems can be expressed in the form of in the recursive form and that helps in writing a very second and tight code.

(Refer Slide Time: 15:26)



- Examples:
  - Factorial:
    - fact(0) = 1
    - fact(n) = n * fact(n-1), if n > 0
  - GCD:
    - gcd (m, m) = m
    - gcd (m, n) = gcd (m-n, n), if m > n
    - gcd (m, n) = gcd (n, n-m), if m < n
  - Fibonacci series (1,1,2,3,5,8,13,21,....)

$f(n) = f(n-1) + f(n-2)$

$f(n-2) = f(n-3) + f(n-4)$

57

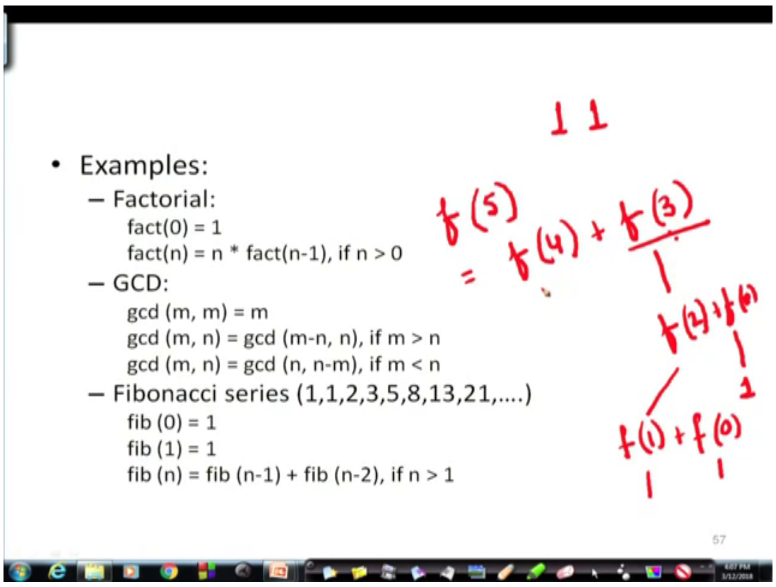
Here is another example of recursion Fibonacci series a series like this one, then one, then 2, 2 is what the sum of the previous 2 elements 1 plus 1 and 3, what is 3? 3 is the sum of the previous 2; 2 and 1, then 5, what is 5? 5 is the sum of the previous two.

Then 8; what is 8? 8 is the sum of the previous 2. 5 and 3, then 13, what is 13? 13 is the sum of the previous 2, 8 and 5, then 21; what is 21? 21 is the sum of the previous 2, 13 and 8, you can see how nice pattern, it is can you think of how we can write it in a recursive form, how we can express it in a recursive form, if you think a little bit, it will be very easy, it will be something like Fibonacci number of n is Fibonacci of n minus 1 plus Fibonacci n minus 2, right.

So, n 21 this one is the sum of the earlier 2 Fibonacci sequences. Now f n minus 2 will be what will be f n minus 3 plus f n minus 4. Similarly, f n minus 1 will be fn minus 1 f n minus 2 plus fn minus 3 in that it will go on, but when will it stop? The stopping condition is that Fibonacci of 1 is one we will see that.

So, if we try to express it in the recursive form, it turns out to be Fibonacci of 0 is 1.

(Refer Slide Time: 17:48)

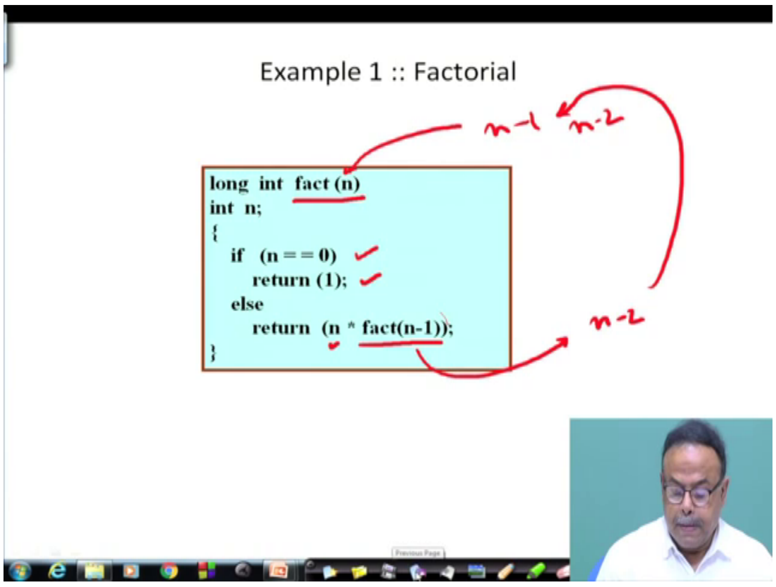


- Examples:
  - Factorial:
    - fact(0) = 1
    - fact(n) = n * fact(n-1), if n > 0
  - GCD:
    - gcd (m, m) = m
    - gcd (m, n) = gcd (m-n, n), if m > n
    - gcd (m, n) = gcd (n, n-m), if m < n
  - Fibonacci series (1,1,2,3,5,8,13,21,….)
    - fib (0) = 1
    - fib (1) = 1
    - fib (n) = fib (n-1) + fib (n-2), if n > 1

That means the first element is one Fibonacci of 1 is 1. There is a second element, then Fibonacci of n is Fibonacci of n minus 1 plus Fibonacci of n minus 2. So, now, suppose I give you Fibonacci of 5, how do you write that it will be Fibonacci of 5 will be Fibonacci of 4 plus Fibonacci of 3 and Fibonacci of 3 will be Fibonacci of 2 plus Fibonacci of 1 and Fibonacci of 1 we know is 1.

So, I know this and then Fibonacci of 2 will be Fibonacci of 1 plus Fibonacci of 0, I know that this is 1, this is 1. So, I get this; this is done. Similarly, if you one X, once this is done I got this number. So, what would that be this will be 2 and so third one will be 2, then Fibonacci of 4 will be 4 plus 3, 2 plus 3, 5; in that it will go on, alright. So, if we try to run, now we see; how we can write a function, how we can express this Fibonacci or this factorial the record recursive expression in the form of a function.

(Refer Slide Time: 19:36)



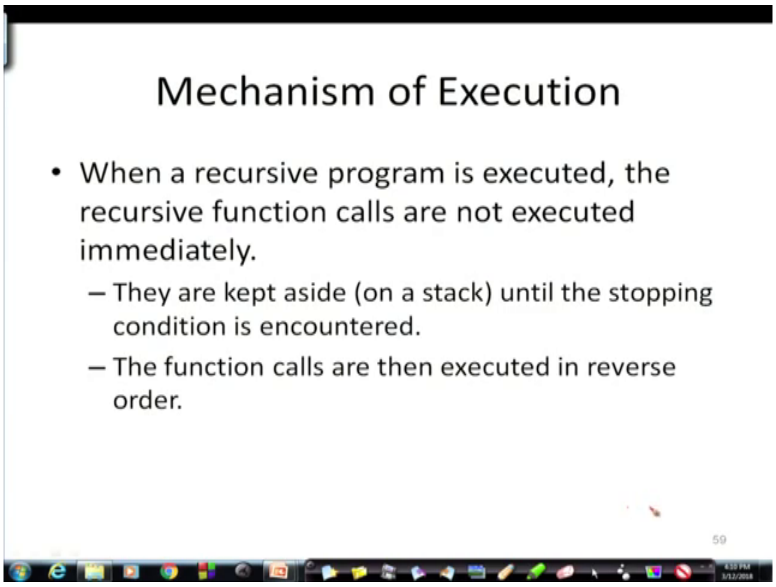
Example 1 :: Factorial

```
long int fact (n)
int n;
{
  if (n == 0)
    return (1);
  else
    return (n * fact(n-1));
}
```

Let us see here, I am writing this, you had seen earlier functions written for factorial. So, you can see factorial n is if n equal to 0, return 1, otherwise return n times factorial n minus 1. So, what will happen? How will this be executed? What will it return while returning, it will again call this function, again, this function will start in the same way just by replacing n with n minus 1 and here, it will come out with n minus 2. So, it will again be called with n minus 2 and so on and so forth.

It will go on, all right, now how is that executed the function as I said is not executed.

(Refer Slide Time: 20:35)



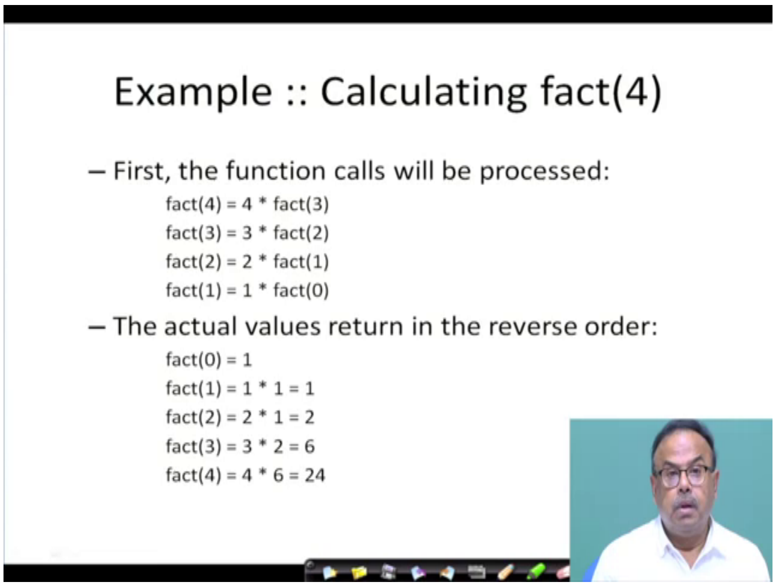
Mechanism of Execution

- When a recursive program is executed, the recursive function calls are not executed immediately.
  - They are kept aside (on a stack) until the stopping condition is encountered.
  - The function calls are then executed in reverse order.

Immediately just as when I asked that coke machine to deliver 5 cokes, he did not deliver immediately, it could deliver 1 coke held it back, but created another machine to deliver n minus 1 cokes and that machine held it back and generated another machine to deliver n minus 2 cokes. In this way, it went on they are kept aside on a stack on a stack one after another until the stopping condition is encountered. So, it remembered that I have to deliver one.

So, if you look at this I do not know whether that will visible or not here here you see everybody remembered that I have to deliver, but they did not deliver when the stopping condition was met after that this back chain started ok. So, they are kept aside, but not delivered immediately the function calls are then executed in reverse order again you can see that they are executed in reverse order here they are executed in the reverse order here in order to get the solution right.

(Refer Slide Time: 22:14)



Example :: Calculating fact(4)

– First, the function calls will be processed:

fact(4) = 4 * fact(3)
fact(3) = 3 * fact(2)
fact(2) = 2 * fact(1)
fact(1) = 1 * fact(0)

– The actual values return in the reverse order:

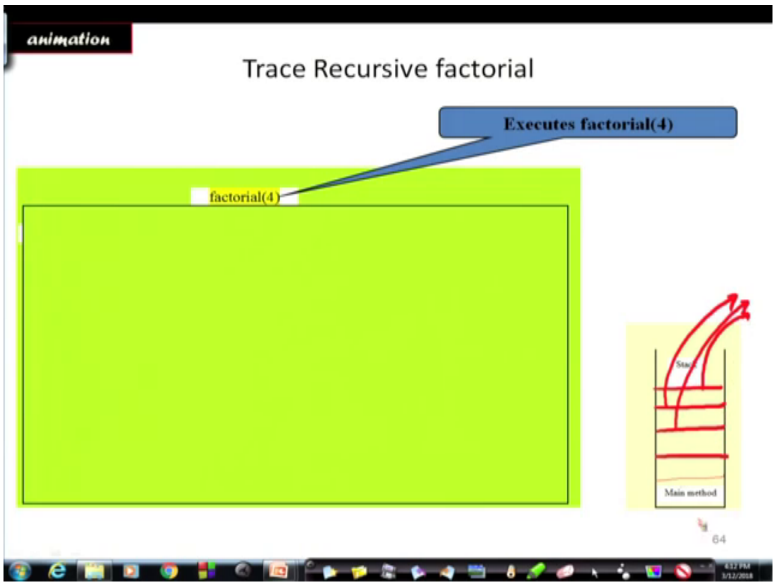
fact(0) = 1
fact(1) = 1 * 1 = 1
fact(2) = 2 * 1 = 2
fact(3) = 3 * 2 = 6
fact(4) = 4 * 6 = 24

Say calculating factorial 4, first the function calls will be processed factorial 4 is factorial 4 times factorial 3, then factorial 3 is 3 times factorial 2 factorial 2 is 2 times factorial 1; factorial 1 is 1 times factorial 0 and factorial 0 is 1 therefore.

Now, the actual values will return in the reverse order 1 into 1 1. So, it fact 1 is complete. So, 1 into 2 is 2 that goes here fact 2 is 2 in that way, it goes on. So, it goes back in this direction this direction this direction and this direction ultimately, we get the result from here. So, the actual values return in reverse order. So, factorial 0 is 1 factorial 1 is 1 times

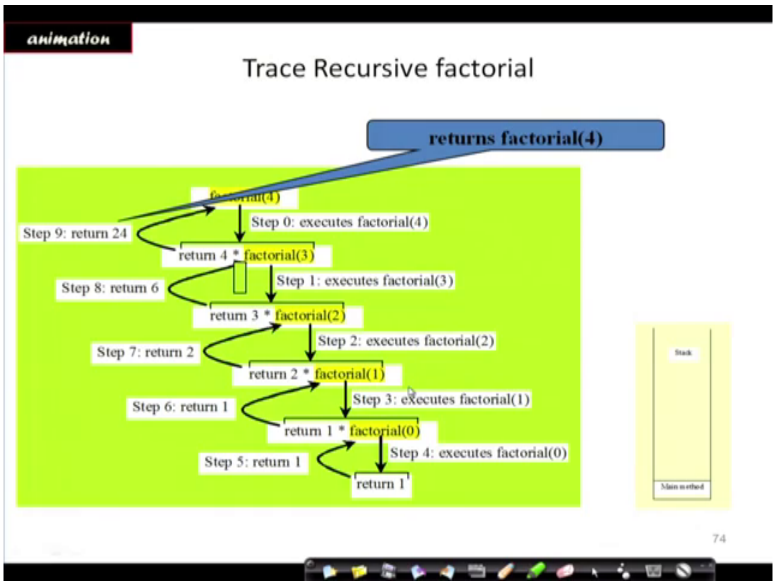
1 1 factorial 2 is now the reverse order is being done 2 into 1 2 3 into 2 6 and 4 into 6, 6; 24 in that way, it is being computed, all right.

(Refer Slide Time: 23:36)



So, if we look at. So, here is a stack. Stack is a data structure stack is a way of storing data where we stored the data just like whatever comes in first; obviously, goes out las last because if I put something here and above that I put something above that I put something just like a stack of books, you cannot you will have to take out in the other way this one will come out first then this one will come out then this one will come out like that all right. So, let us see how it works factorial 4.

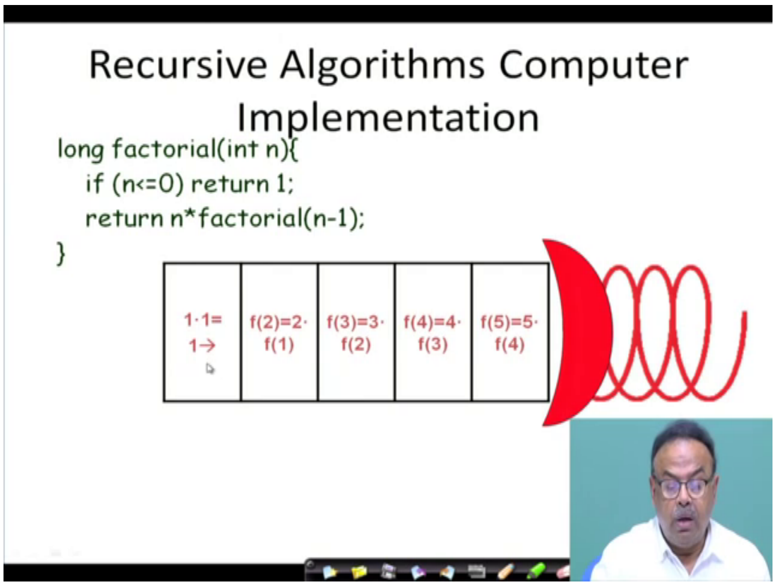
(Refer Slide Time: 24:14)



So, factor step 0 execute factorial 4 that is executes now return 4 times factorial 3, you see here is the recursive call it is calling itself return 4 times factorial 3, what is factorial 3, oh you do not know; what is factorial 3 return 3 times factorial true 2; oh you do not know; what is factorial clue 2 ok, then return 2 into factorial 1 or you still do not know; what is factorial 1 return 1 into factorial 0 and so, now, you know factorial 0 is 1. So, you get 1 and now you could not answer these questions earlier I have broken it down and gave you an easier solution.

And so, now, you go back here and you return one into one then you go back here, return 1 into 2 return 2 and you go back here go back here go back here in that way this is what is meant by recursion if we redo it if we.

(Refer Slide Time: 25:46)



Recursive Algorithms Computer Implementation

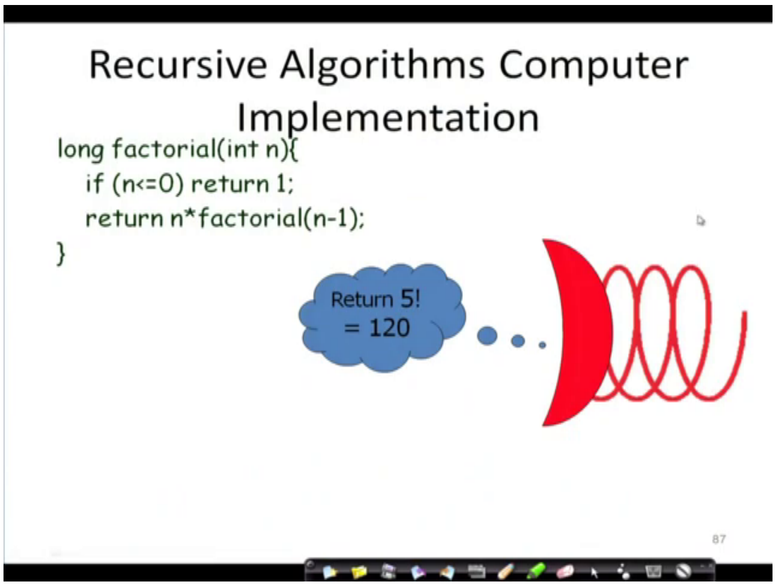
```
long factorial(int n){
   if (n<=0) return 1;
   return n*factorial(n-1);
}
```

And while we are doing this; so, here is computer implementation of that factorial if n is less than 0, return 1, return n times factorial n minus 1 ok.

So, computes 5. So, it is again recursively expressing itself f 4 is being expressed in terms of f 3, everything is being expressed f is being expressed in terms of f f 3 is being expressed in terms of f 2, f 2 is being expressed in terms of f 1; f 1 is being expressed in terms of f 0 and I know what f 0 is. So, f 0 is 1. So, I go back and I know that now f 0 is 1.

Therefore f 1 is 1 into 1 and in this way, I go back in this way. Now, you see it had expanded in this way. Now your it is shrinking 1 into 1 is 1e. So, now, I know factorial 1. So, this will shrink; now I know what is factorial 2 that is 2. So, this will shrink 3 into 2; 6, it is shrinking coming here 24 and then here 120; that is how as if in a spring it got expanded and then it contracted back, this is recursion return it ultimately returns factorial 5 to be 120.

(Refer Slide Time: 27:10)

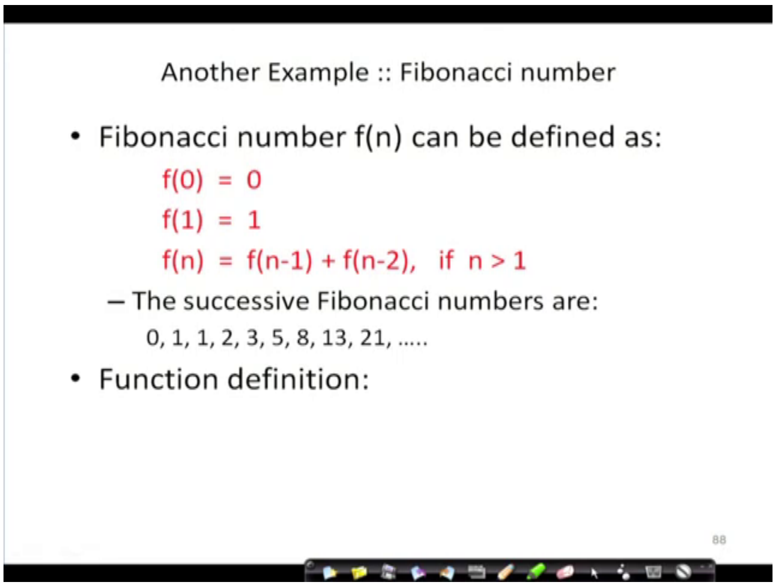


Recursive Algorithms Computer Implementation

```
long factorial(int n){
   if (n<=0) return 1;
   return n*factorial(n-1);
}
```

Return 5!
= 120

(Refer Slide Time: 27:10)



Another Example :: Fibonacci number

- Fibonacci number f(n) can be defined as:

  f(0) = 0

  f(1) = 1

  f(n) = f(n-1) + f(n-2), if n > 1

  – The successive Fibonacci numbers are:

  0, 1, 1, 2, 3, 5, 8, 13, 21, …..
- Function definition:

Another example will be Fibonacci number. We will explain it in the next lecture.