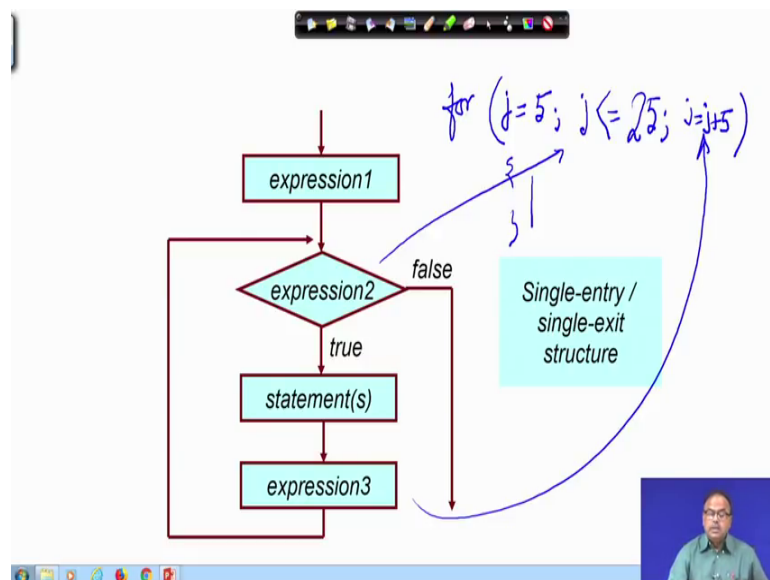


**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 21**  
**For Statement (Contd.)**

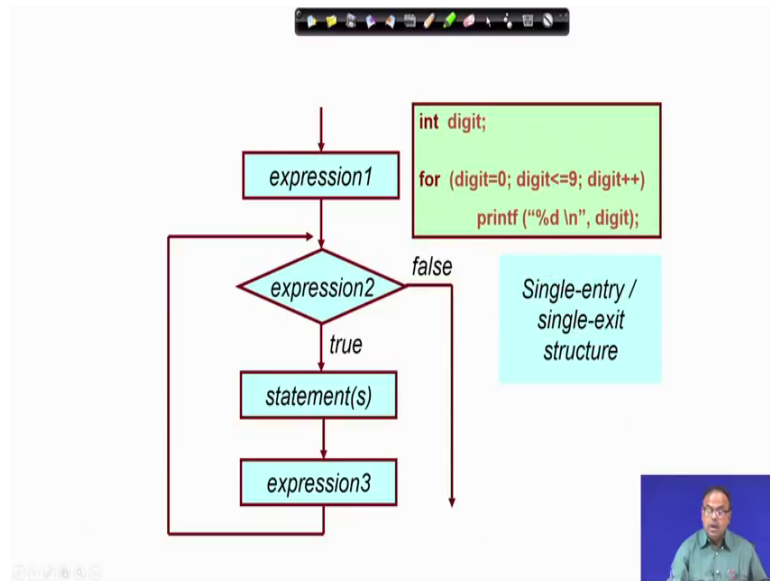
Now, we are discussing about formation of loops, creation of loops in a program using for statement. Before that we have seen the while statement and do while statement and then we have seen the for statement as well.

(Refer Slide Time: 00:35)



Now, the for statement as we had shown we had discussed in the last class it is basically starting to take an expression, arithmetic expression and then it starts with in sorry initialization expression and then we check a particular condition. So, something like this for some variable integer variable  $j$  assigned some value 5 maybe and then some expression here this one is  $j$  less than equal to 25 and then here there will be some statements which will be executed and after that there is a third expression which is altering it and that can be  $j$  assigned  $j$  plus 5 all right and the body of the x loop. These are structure that we had seen.

(Refer Slide Time: 01:52)



Now, there are some critical issues that are to be noted for the for structure.

(Refer Slide Time: 01:53)

### The For Structure: Notes and Observations

- Arithmetic expressions
  - Initialization, loop-continuation, and increment can contain arithmetic expressions.
  - e.g. Let  $x = 2$  and  $y = 10$

```
for ( j = x; j <= 4 * x * y; j += y / x )
```

is equivalent to

$j += y/x$   
 $j = j + y/x$

So, if you look at this you can see that we are using arithmetic expressions. For example,  $x$  assigned 2 and  $y$  assigned 10 that is an arithmetic expression. So, similarly this is a valid arithmetic expression for  $j$  assigned  $x$ ,  $j$  less than equal to 4 times  $x$  times  $y$  that is the condition. That means, after some computation will have to find out whether  $j$  is satisfying this condition and here  $j$  here probably this is something this construct we have not shown you, this is I do not like it very much and initially you need not bother about

this j plus equal to y by x this is a C structure, say C syntax expressing j equals j plus y by x. So, the whole thing can be written in this way.

So, this, but that is not important for the purpose of understanding for expressions. So, here is just an expression.

(Refer Slide Time: 03:32)

The For Structure: Notes and Observations

- Arithmetic expressions
  - Initialization, loop-continuation, and increment can contain arithmetic expressions.
  - e.g. Let  $x = 2$  and  $y = 10$   
`for ( j = x; j <= 4 * x * y; j += y / x )`

is equivalent to

`for ( j = 2; j <= 80; j += 5 )`

- "Increment" may be negative (decrement)

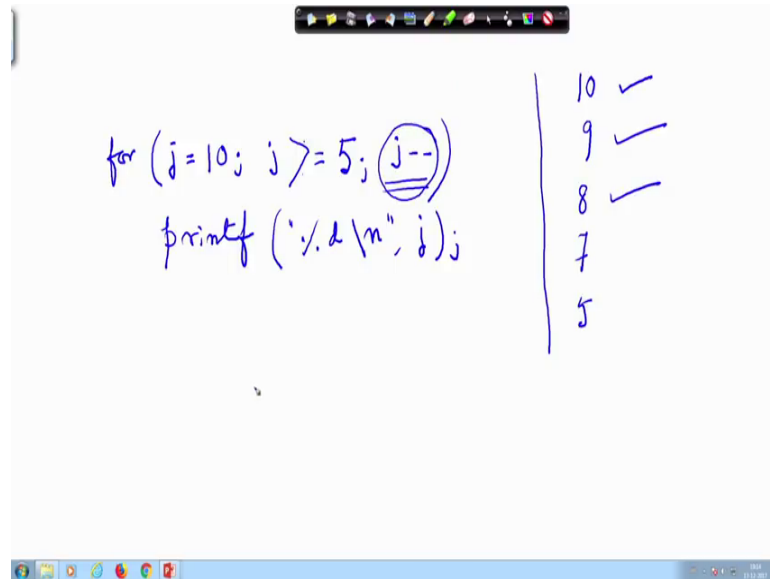
`for ( j = 2; j <= 80; j-- )`

Diagram labels: Initialization, Increment, Loop continuation

And this is equivalent to like this for say x was 2 and y was 10, then this is equivalent to j assigned 2 because j assigned x and j assigned 2 are essentially same; j as less than equal to 4 times x times y that means, 4 times 2 times 10 that means, 80 and j equals assigned j plus y by x is 5. So, this is equivalent all right. So, this is initialization, this is loop continuation condition and this is the increment or I as I was saying that this is a modulator or the alteration.

Now, increment can be negative. So, this increment all the way we are calling it increment that is why we should not call it increment let us call it a modulator because say the same thing I can write as for j assigned 2 maybe something j less than equal to 80 and might be j minus minus. It can be the case when can it be. Say for example, if I add if I want that I will be printing the numbers in the reverse order starting from 10. So, what should I do? So, can I do this what I want to do is, that I want to print something like 10 9 8 7 5 all right in this order.

(Refer Slide Time: 05:15)



The image shows a whiteboard with handwritten code and its output. The code is written in blue ink and consists of two lines: a for loop and a printf statement. The for loop is `for (j = 10; j >= 5; j--)` and the printf statement is `printf ("%d\n", j);`. To the right of the code, there is a vertical line with the numbers 10, 9, 8, 7, and 5 listed vertically. Next to each number is a checkmark, indicating that these values are printed by the program. The whiteboard also shows a Windows taskbar at the bottom and a toolbar at the top.

```
for (j = 10; j >= 5; j--)  
printf ("%d\n", j);
```

10 ✓  
9 ✓  
8 ✓  
7  
5

So, simply I can do this repeatedly in a loop for example, for j assigned 10 and j sorry, j greater than equal to 5 j minus minus and here I just do printf percentage d backslash n j. So, what will happen? Initially j is 10 less greater than equal to 5, so 10 will be printed. Then I go back I decrement j. So, j will become 9 still greater than equal to 5 I come in here print j 9, I again decrement j decrement j becomes 8 I compare with this still it is greater than equal to 5 I get in and printed. In that way I can do it I can repeatedly do the same thing, but here as you can see I am being able to achieve this reverse order by instead of incrementing and decrementing this index. That is why I can increment to decrement, I can multiply it, I change it, I modulate it, that is why this to an alteration expression or modulation expression is a better term than increment or decrement.

(Refer Slide Time: 07:31)

**The For Structure: Notes and Observations**

- Arithmetic expressions
  - Initialization, loop-continuation, and increment can contain arithmetic expressions.
  - e.g. Let  $x = 2$  and  $y = 10$   
`for ( j = x; j <= 4 * x * y; j += y / x )`

is equivalent to

```
for ( j = 2; j <= 80; j += 5 )
```

Diagram illustrating the components of a for loop:

- Initialization**: `j = x`
- Loop continuation**: `j <= 4 * x * y`
- Increment**: `j += y / x`

- "Increment" may be negative (decrement)
- If loop continuation condition initially false
  - Body of for structure not performed
  - Control proceeds with statement after for structure

41

So, if the loop continuation, if the loop continuation condition is initially false is initially false then the body struck the for structure will not be executed. It will proceed with the statement for the next statement.


(Refer Slide Time: 07:57)

Handwritten code example:

```
j = 20
for ( i = 1; i > j; i++ )
{
}
//
```

Diagram illustrating the components of a for loop:

- Initialization**: `i = 1`
- Loop continuation**: `i > j`
- Increment**: `i++`



So, for example, if I write something like this as you have seen earlier  $j$  equal to 20 and for  $i$  equals  $i$  assigned 1,  $i$  greater than  $j$   $i$  plus plus say and then I want to do some things here all right.

Now, j is 20 I will first initialize i to 1 and immediately I find the first thing that I do is its false I check this and its false i is not greater than j therefore, this stay these statements will not be executed even once and the statements following the for loop will be executed.

(Refer Slide Time: 08:55)

The slide is titled "for :: Examples". It contains a code block with the following C++ code:

```
int fact = 1, i;  
for (i=1; i<=10; i++)  
    fact = fact * i;
```

Below the code, there is a handwritten diagram illustrating the loop's execution. It shows the variable 'fact' starting at 1. As 'i' increases from 1 to 4, the value of 'fact' is updated to 'i \* fact'. The diagram shows the sequence of calculations: 1, 1.2, 1.2.3, and 1.2.3.4. The final result for i=4 is 1.2.3.4.

The slide also features a small video inset of a person in the bottom right corner.

Now, here are some examples of for loops. This is again computing the factorial fact. Factorial is a the variable, fact is touring the factorial factorial all of you know as factorial n is n multiplied by n minus 1 multiplied by n minus 2 so on so forth up to 1.

So, it is 1 and I have got a variable i. Now, you can see simply I can do it like this i assigned 1, i less than 10 i plus plus fact assign fact times i. Now, what is happening here? So, I have got fact to be 1. So, i equals 1 then with that what I do is 1 times 1 is fine and I then make i to be 2 and then i, i becomes then 2. So, 1 times 2 then i becomes 3 then 1 times 2 times 3 then i becomes 4 then 1 times 2 times 4 times 3 times 4 and so on and so forth it will go on in this loop. So, that is a nice way of writing factorial.

(Refer Slide Time: 10:27)

### for :: Examples

```
int fact = 1, i;
for (i=1; i<=10; i++)
    fact = fact * i;
```

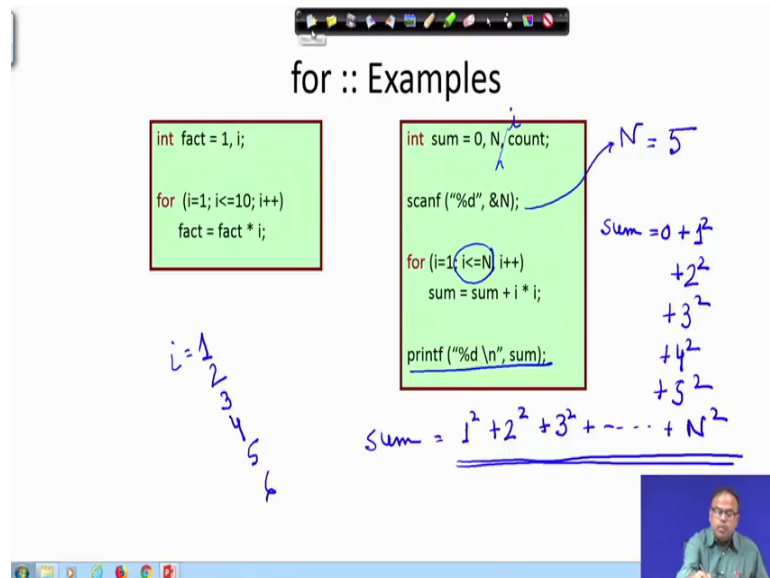
$i = 1, 2, 3, \dots, 5, \dots$

```
int sum = 0, N, count;
scanf ("%d", &N);
for (i=1; i<=N; i++)
    sum = sum + i * i;
printf ("%d\n", sum);
```

$N = 5$

$sum = 0 + 1^2 + 2^2 + 3^2 + 4^2 + 5^2$

$sum = 1^2 + 2^2 + 3^2 + \dots + N^2$



And here is another example you can see what it does quickly. You can explain it yourself what it does. Sum is 0 and N is a variable and count is another integer variable all right. Sum is 0, N is a 1 variable and count is another integer variable. Now, we are reading N and for i equals 1, i less than equal to N I am adding some. So, what is being done here what will happen if I press this program what is going to happen? Let us do that.

Suppose I have read N to be here, I have read N to be 5. Now, what am I doing here? I here there is one problem here i should have been declared there is a mistake here int N and here i also add i should have been declared here. Now, i assigned 1. So, i is 1, N was 5, i less than 5 less than equal to 5. So, sum is sum was 0. So, sum is 0 plus i times i, i was 1. So, 0 plus 1 square and then I check i increment i. So, i becomes 2, i becomes 2 and I check that i is less than it not less than a is still less than equal to 5 so I got this sum and with that I add now, i square i times i, so 2 square. Similarly, it will go on it will do 3 square. So, i will become then 4 and then still it is less than equal to, so 4 square then i will become 5 and still it is true. So, it will be 5 squared then i become 6 this condition will be violated and i will come to this printf.

So, what will sum be sum is 1 square plus 2 square plus 3 square and if I make it N then up to N square. So, this is the very well known series the sum of the square of natural

numbers we can compute by this small program using for loop right. So, it is a nice example.

(Refer Slide Time: 13:32)

- The comma operator
  - We can give several statements separated by commas in place of “expression1”, “expression2”, and “expression3”.

```
for (fact=1, i=1; i<=10; i++)  
    fact = fact * i;
```

```
for (sum=0, i=1; i<=N; i++)  
    sum = sum + i * i;
```

for (i=1; i<=N; i++)

Now, we introduce the comma operator. As we say that here for when we write for then i assigned 1 instead of that I can put in more than one statements here using a comma operator. For example, for fact 1 i equals 1, i less than equal to 10. Now, this part is what? This part is the initialization. Now, remember that this will not be continuously initialized, this is just an just a statement that I put here an assignment statement, but my index variable for the control variable for the loop is i. So, I could have written this earlier example of factorial maybe, earlier example of factorial this could be initialized here also all right. So, that is just saving space saving the number of lines of code.

But personally I would suggest as I did earlier also that is for of those of you who are beginners in programming you should not try this tricks or should not try this thriftiness reducing the number of lines, that is not so important to how much you can reduce. The most important thing is to be logically can syntactically correct while you write a program ok.

So, we can give several things in comma like some here again. So, the program becomes even smaller look smarter, but sometimes at the beginning if you try to do that you try to be smart and in the process you may result in some long program long long logic better avoid that.



(Refer Slide Time: 15:27)

The slide is titled "Specifying 'Infinite Loop'". It features a handwritten C++ code snippet: `for (i = 1; i < N; i++) { i--; }`. The condition `i < N` is circled in blue. A blue arrow starts from the `i++` part of the loop, goes down, then left, then up, and finally loops back to the `i < N` condition, illustrating that the condition is never satisfied. To the right of the code is a simple diagram of an oval with an arrow pointing clockwise, representing an infinite loop. The slide also shows a Windows taskbar at the bottom and a small video inset of a presenter in the bottom right corner.

Now, infinite loop in general what is an infinite loop. When a program continues, program continues in a loop repeatedly it goes on and it is never completing its going on because the condition that is supposed to turn out to be false at a particular point of time. So, that it comes up the out of the loop never happens all right. That can always happen that say if I write something like for i equals 1, i less than n and whatever i plus plus. Now, every time inside the loop when you get in you read you do something and then increment i and when you come inside the loop you do i minus minus, then whatever has been done here will be canceled out here. So, this loop will never reach this condition I less than i therefore, that this loop will continue forever such situations are known as the condition of infinite loop.

Now, sometimes usually we do not like that, but sometimes it may be necessary to specify that something will happen forever all right. Say some particular work has to be done continuously.

(Refer Slide Time: 07:23)

The slide is titled "Specifying 'Infinite Loop'". It features three code snippets in green boxes:

- ```
while (1) {  
  statements  
}
```
- ```
for (;;)  
{  
  statements  
}
```
- ```
do {  
  statements  
} while(1);
```

Handwritten blue annotations are present:

- A large blue bracket on the left side of the do-while loop.
- Handwritten text  $i = 5$  next to the opening brace of the do-while loop.
- Handwritten text  $i \leq N; i--$  next to the while(1) condition.

A small number "44" is visible in the bottom right corner of the slide.

Now, in order to specify that for loop provides us some facility like say for and while everything say this one we had discussed earlier while 1, that means, always it is true it is a nonzero this while loop this expression part condition expression part should return nonzero and so it will go on. Now, here for and I put null; that means, the for has had 3 parts, for the initialization part, some condition part and some incrementation decrementation part whatever.

Now, I just keep everything blank. So, I turn it to be for nothing and nothing all these things are blank. In that case what will happen? In that case what will happen? If I keep some statements here that will go on forever. Similarly if I in the case of do while if I just put while 1 that is again in finite loop. So, by this I can express my desire that some things will have should continue forever all right. So, this is another trick that you can utilize in some cases for the for and the while constraints.

(Refer Slide Time: 19:05)

**break Statement**

- Break out of the loop {}
  - can use with
    - while
    - do while
    - for
    - switch
  - does not work with
    - if {}
    - else {}

**Causes immediate exit from a while, for, do/while or switch structure**

**Program execution continues with the first statement after the structure**

**Common uses of the break statement**

- Escape early from a loop
- Skip the remainder of a switch structure

*Handwritten notes:*

- Break out of the loop.
- switch color { case 'R' : break; case 'G' : break; }

Now, we come to another statement which we have encountered a little few lectures earlier in the context of switch statements. While we are considering the switch statements if you recall you have seen that after every case statement switch on a particular variable then their case red case, green case, blue if you recall then we did something and then give a break, did something give a break like that we proceeded right.

So, the break statement of course, we know can help us we can use it with several things one is while, do while, for and switch. Now, this switch, switch part we have seen, but we can also use it for while do while and for, for breaking out of the loop, we want to break out of the loop. Sometimes that is required it works with while, do while, for, but does not work with if and else statement. It causes immediately exit from a while for a do while or case we have seen that switch statements earlier.

So, the program continues after with the next statement let us see. Why do we want to use it? It helps us to escape early from a loop. Sometimes we want to escape early I do not want to go till the end of the loop, when a particular condition is met I want to come out. I am doing it in a loop, but waiting for some condition to take place as soon as that condition takes place I come out of the loop. Maybe, as we have seen in the case statement switch some expression all right color, then case R we do something and then

give break right, case G we do something and then give break right we do like that. So, we skip the remaining part of the switch block right. We come out of that.

(Refer Slide Time: 21:54)

### A Complete Example

```
#include <stdio.h>
main()
{
    int fact, i;
    fact = 1; i = 1;
    while (i < 10) {           /* run loop - break when fact > 100 */
        fact = fact * i;
        if (fact > 100) {
            printf ("Factorial of %d above 100", i);
            break;           /* break out of the while loop */
        }
        i++;
    }
}
```

*Handwritten annotations:*

- Red curly braces on the left side of the code block, grouping the initialization, the while loop, and the closing braces.
- Handwritten numbers below the code: "5, 4, 3, 2" under the while loop and "1 2 6 24 (120)" under the if statement.
- A blue circle around the value 120 in the sequence.

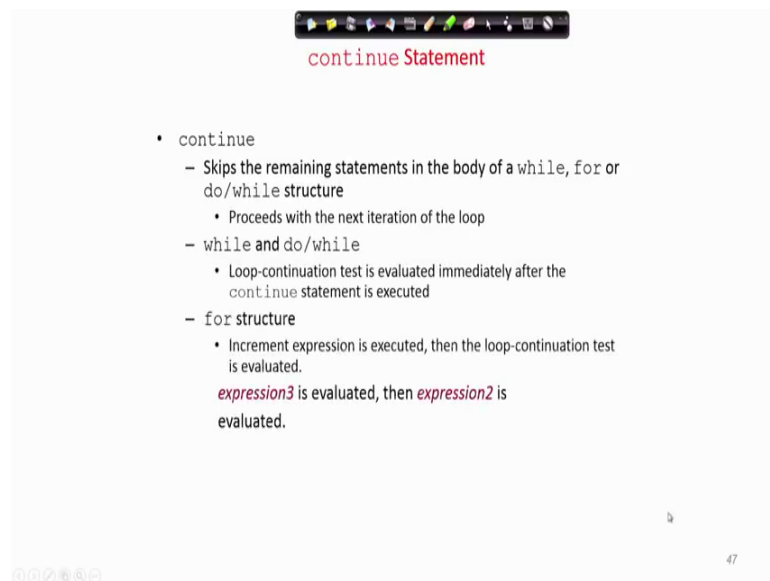
So, similarly we will see a couple of examples of this. Say here there is a complete example let us look at that. Here is a complete example complete program, once again include stdio dot h you are running some programs main, then fact and i are integers fact we are computing factorial. So, fact is 1, i is 1, while i is less than 10; that means, I want to break out while i is less than 10 fact times i, if fact is greater than 100, then factorial is above 100 factorial of a number is above 100. So, for example, I am going on up to 10 numbers. So, what will happen with the factorials? Let us see. So, factorial 1 will be 1, factorial 2 will be 2, factorial 3 will be 3 times 2 that is 6, factorial 4 will be 4 times 3 for 4 times 6, it will be 4 times 3 times 2, so 4 times 6; that means 24, then 24 times 5, so that will be how much it will be, more than how much will it be if I have got this 4 3 2 and multiply that with 5, so 26.

So, here just with 5 numbers I am exceeding the value 120, but I do not know when I am going to reach the value 100. So, I have written the program in this way, fact while i is less than 10 I will test it up to 10 numbers every time, I compute fact if fact is greater than 100 print. So, here as soon as fact becomes 100 will say factorial of 5 is above 100, factorial of 5 is above 100 and then I break out. Otherwise if I had not put this break then this would have continued till this loop all for all the 10 numbers up to factorial 10, but I

just want to stop whenever my result becomes more than 100. So, this is one way we can utilize a break right.

Similarly there is another statement although we do not use it often it is better to know that sometimes it can come handy, that is a continued statement; that means, the remaining part of the body of a loop we will skip if I put a continue all right.

(Refer Slide Time: 25:05)

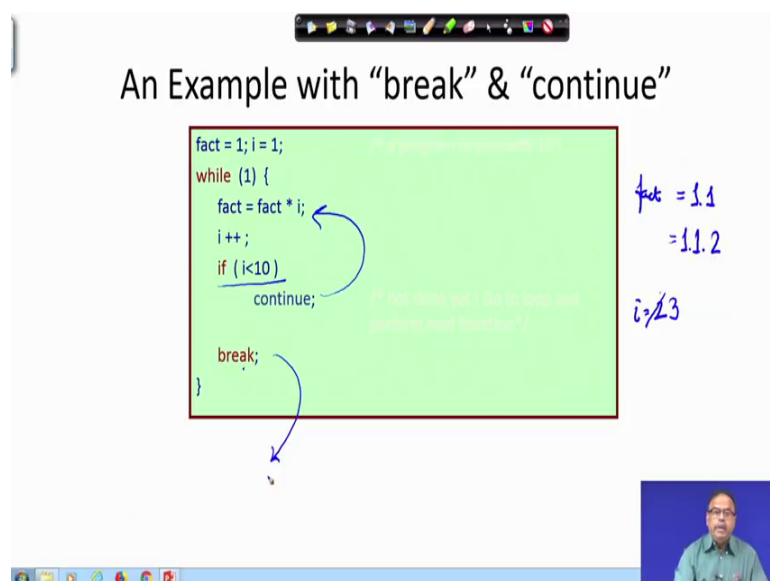


**continue Statement**

- continue
  - Skips the remaining statements in the body of a while, for or do/while structure
    - Proceeds with the next iteration of the loop
  - while and do/while
    - Loop-continuation test is evaluated immediately after the continue statement is executed
  - for structure
    - Increment expression is executed, then the loop-continuation test is evaluated.  
*expression3* is evaluated, then *expression2* is evaluated.

So, let us see and it proceeds to the next iteration of the loop. So, what happens is let us look at one example here.

(Refer Slide Time: 25:11)



**An Example with "break" & "continue"**

```
fact = 1; i = 1;
while (1) {
    fact = fact * i;
    i++;
    if (i < 10)
        continue;
    break;
}
```

*/\* not there yet. Go to loop and perform next iteration.\*/*

fact = 1.1  
= 1.1.2  
i = 3

*(Note: A small video inset of a speaker is visible in the bottom right corner of the slide.)*

So, while one that means what; that means, it is always true. It will continually go on doing it. So, fact is fact times  $i$ ,  $i$  plus plus. So, I start with  $i$  1, then 2, then 3. If  $i$  is less than 10 I will go into the loop otherwise I will break. So, this continue is basically remain forcing me to go back to this point, by forsaking this part. I am not coming to the remaining of the loop, but if this condition is not holding then I will not execute continue I straight away come here and do break. I think it will need a couple of moment for you to just realize what is happening here all right.

So, just look at this. So, in this way it will I think you can understand it. So, I have got  $i$  to be 1. So, fact will be 1 times 1, then  $i$  becomes 2 and then I go back. So, it will be fact will be 1 times 1 times 2,  $i$  will be 3 and in that will go on, but as it will go on as long as  $i$  is 10, less than 10. Then  $i$  will not come to break I will go back here. But if  $i$  is less than 10  $i$  will come to break and  $i$  will come out of the loop. So, it is a combination of continue and break this is. But what I want is that you should understand how the for loop is used for repetition that is very important to understand and will carry out with some examples in the next lecture.