Introduction to Problem Solving and Programming
Prof. Deepak Gupta

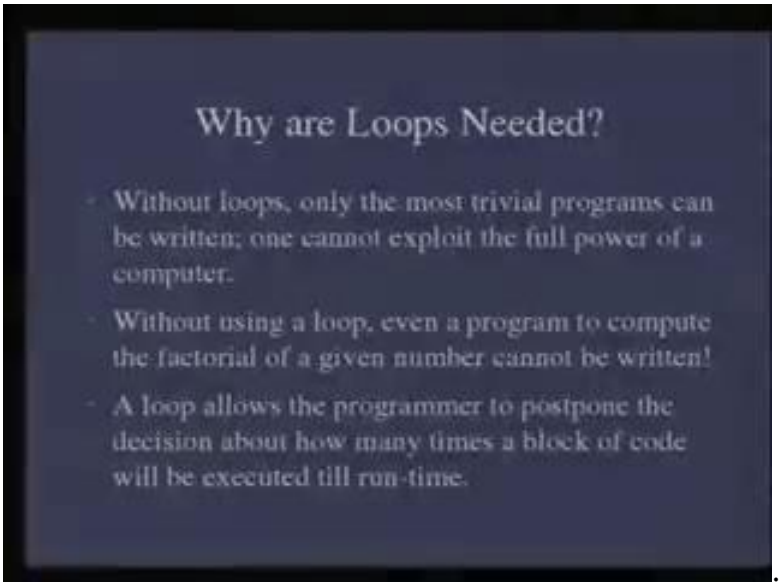
Department of Computer Science Engineering

Indian Institute of Technology, Kanpur

Lecture - 11

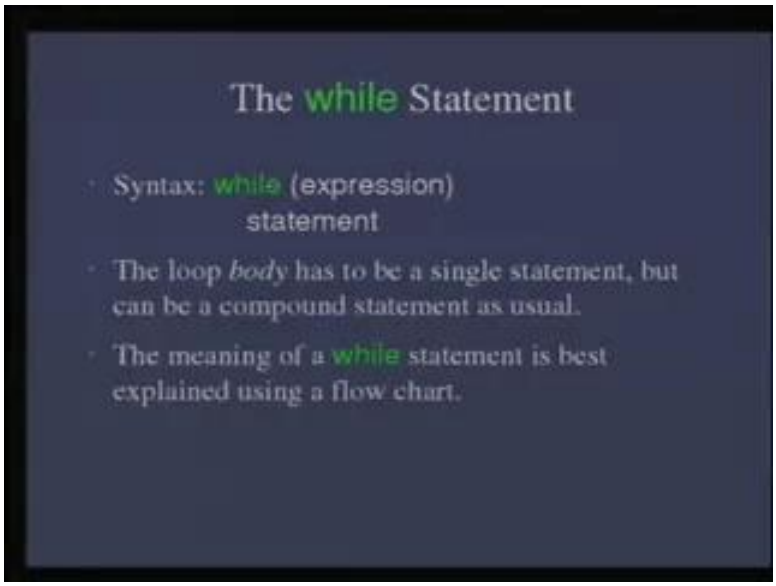
In the last lecture, we had started looking at the control statements in C, and we had talked about the if, if else and the switch statements. Today, we will look at another very important class of control source statements or control statements, which are the statements for implementing loops.

(Refer Slide Time: 00:28)



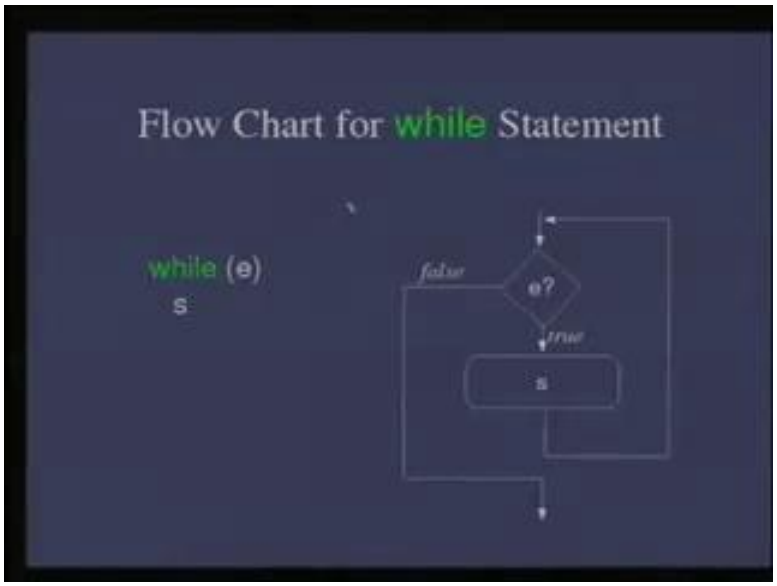
We already used loops in most of the programs that we have written, and by now we should be already aware that without loops only the most trivial programs can be written. We need to in many situation postponed the decision about how many times a certain piece of code will execute till that time that is based on the num based on the input feed by the program. Sometimes we may need to decide how many times a particular piece of code should execute. And for doing that of course, we need a loop; without a loop only what we can do is if it is known in advance how many times a piece of code is to be executed then we can replicate that code that many times. But without using a loop, we cannot postponed that decision to the run time or in the other words base the decision on the inputs that the program has received.

(Refer Slide Time: 01:15)



Whereas the first statement for implementing the loop that is the while statement, we have already used as in many programs that we have written so far. The syntax is very simple while and then we have an expression in brackets followed by a single statement. So, this statement again just like if and if else has to be single statement. But if you want to have multiple statement here, we can enclose them in braces and form a single compound statement. So, the meaning of this loop is that the expression is the evaluated and the if it is true, the statement is executed; and again the expression is evaluated if it is still true the statement is executed again and so and so forth, and this process continues till the evaluation of expression leads to the value false. We should thus explain using a flowchart, which we have in the next slide.

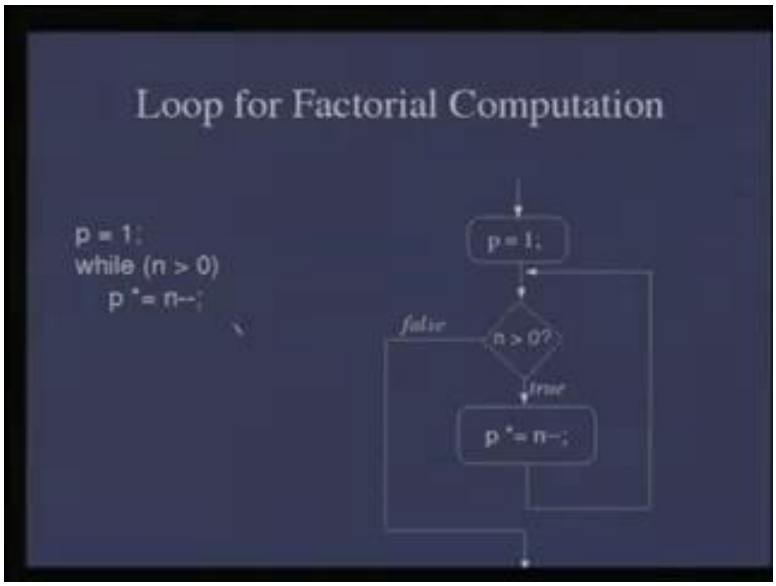
(Refer Slide Time: 02:05)



So, this diagram represents the flowchart for the loop shown on the left hand side, which is while e s, here e is the loop condition and s is known as the body of the loop. So, in this flow chart, the execution begins here and the first thing that happens here is that e is evaluated, the expression e is evaluated. Note that again the expression e may have side effects. So, evaluating the expression will mean that those side effects do certainly take place. And finally, the evaluation of e results in a value, and depending on whether that is true or false, again you will recall that in C any nonzero value is taken as true and zero is taken as false. So, if the value turns out to be true, that is any non zero value then s is executed; and after execution of s, the control goes back to the beginning of the loop and e is evaluated again and this process continues.

When after sometimes, when e evaluates to false at that time the control takes this path and goes to the statement following the while loop. Note that it is important to understand that e may evaluate to false, the first time around itself; if that happens then the body of the loop s will not gets executed even one.

(Refer Slide Time: 03:29)



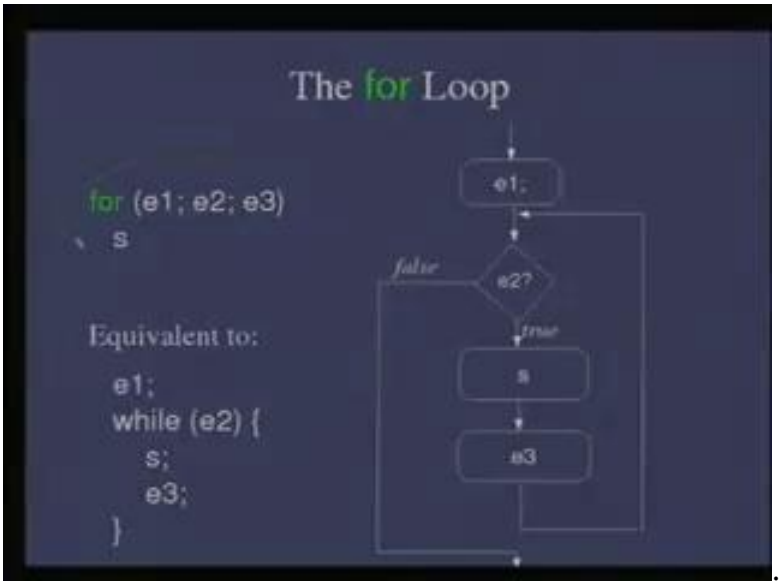
Lets now see some examples of using the while statement using familiar programs. So, here is the while loop for computing the factorial. We initialize p to one and while n greater than zero p star is equal to n minus minus, there is a post decrement of that we are already similar with and this star equal to as you know is equivalent to p assigned star equal to n minus minus. So, the control flow or the flowchart for this particular piece of code would look like the diagram shown in the right hand side of the slide.

So, we first compute or execute the statement p assigned one note that this is outside loop body. So, this will be executed only once and then the loop execution starts. In the loop execution we test n greater than zero if yes execute p star equal to n minus minus and then go back and test the loop condition again. Let us try to trace this loop for some initial value of n, let us assume that initial value of n happens to be two. So, p assigned one, and we test whether n is greater than zero, it is because n is two and this point of time. So, answer is true and the control goes to this statement p star is equal to n minus minus, what will that do that will multiply p by two which is the value of n minus minus remember the initial value of n was two. So, p will become one times two which is two and n because of the post decrement will become one then we go back to the loop. We test again whether n is greater than zero yes indeed it is because n is one now, so that evaluates to true.

So, we again do this, so p becomes two times one which is two and n becomes zero and then we go back to the loop and test again whether n is greater than zero. Now, n has become zero. So, this condition turns out to be false. So, therefore, we come out of the loop and value of p as you know at the end of the loop is the value of the factorial of n. Note that this loop will work correctly even if the value of n initially is zero. As you know by definition zero factorial is equal to one. So, let us lets again look at this flowchart and see what will happen for n is equal to zero.

For n is equal to zero, p will get assigned to one as usual; and then when this loop condition is tested for the first time, it will evaluate to false, because n is equal to zero which is not greater than zero. So, the loop body which is p star is equal to n minus minus does not executed gets even ones, and the controls goes out of the loop immediately and the value of p remains one is of course the correct answer for factorial of zero.

(Refer Slide Time: 06:15)



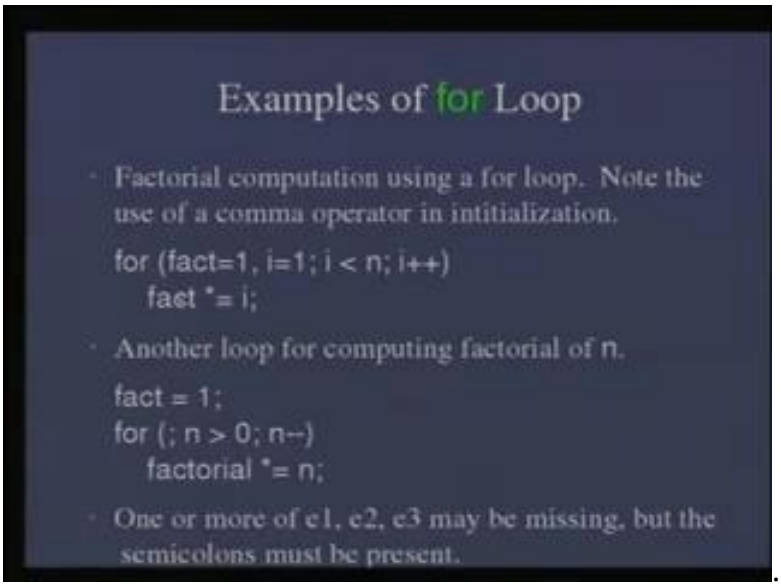
Let us look at another very commonly used kind of loop construct that is a for loop and the syntax is shown here, for is the keyword and within this a within bracket there are three expressions e 1, e 2, and e 3 which have to be separated by a semicolon. Now, e 1 is the initialization this is what will happen once at the beginning of the loop. The expression e 2 is evaluated every time as the loop test that is the loop will continue to run as long as the value of e 2 turns out to be true. And e 3 is step that is done at the end of

every loop body execution that is whenever s is executed after that e 3 is evaluated and then the control goes back to evaluation of the loop test.

So, the flowchart for the loop shown here looks like the one shown on the right hand side of the slide e 1 remember is the initialization. So, this e 1 is an expression which is evaluated only for its side effects for the simple reason that its value that is the value of the expression e 1 is discarded. So, the expression e 1 would usually has some side effects. So, e 1 is the initialization. So, that happens once and after that the loop condition which is e 2 is evaluated that again may of course, have side effects and if it happens to be true then the body of the loop s is executed followed by an evaluation of the expression e 3.

Now, again the value of e 3 the resulting value of e 3 is discarded. So, e 3 also would usually be an expression with some side effect. And once e 3 is evaluated the control goes back to the loop condition note that e 1 is done just one. So, control goes back and evaluates e 2 again. When e 2 evaluates to false the control goes out of the loop and executes the statement after the loop. So, you can see that this for loop that we have written is entirely equivalent to this while loop. So, imagine that we execute e 1 first as the initialization and then e 2 is the loop condition and then e 3 is something that we execute at the end of s. Now, the reason this e 3 is important is because many times when we are looping over numbers or when when we want to execute a loop a certain given number of times then we need to increment a counter in every loop iteration. So, those kinds of loops are very neatly captured or implemented using the for loop.

(Refer Slide Time: 08:59)



Let us see some example. So, here is a loop for computing the factorial again, but this time written using the 'for' statement. So, the e 1 which is initialization is in this case the entire expression set the kind one comma i is equal to one or i assigned one. So, this comma. In fact, is another operator that we have not seen so far. When we have an expression like e 1 comma e 2 as in this case e 1 here is set equal to one and e 2 is i assigned one. So, when we have an expression e 1 comma e 2 what happens is that e 1 is evaluated first and its value is discarded then e 2 is evaluated and its value is the value of the overall expression.

In this particular case, of course, the value of the overall expression is being discarded. So, that is not really material in this case. So, what will happen is that when this expression is evaluated, there will be two side effects fact will be assigned to one followed by i being assigned to one. And then this is the expression which represents the loop test i is less than n and then this is the expression that is evaluated at the end of the loop body.

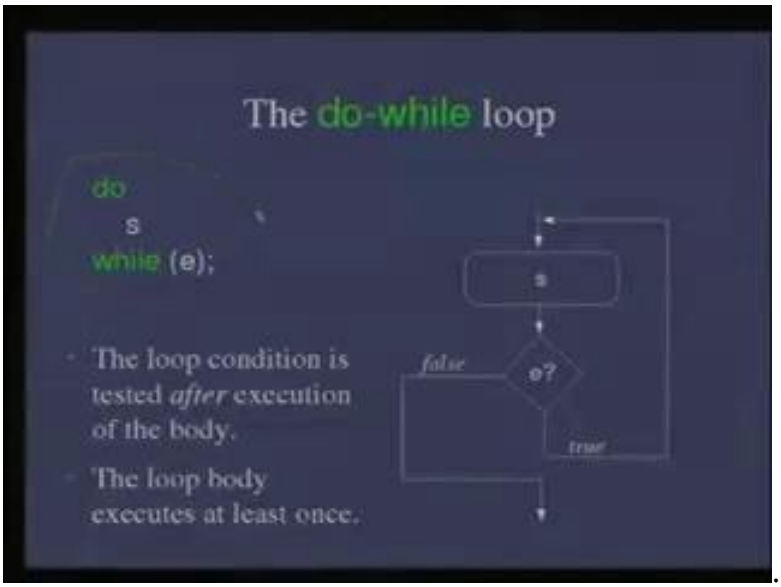
So, what is happening here is that we are instead of multiplying numbers from n to one in the reverse direction in the decreasing order we are now multiplying numbers from one to n in the increasing order and we are not modifying n. Instead of that we are starting with the i value is equal to one and we are incrementing it every time. So, that at the end of the loop it will become n and the loop body is the single assignment statement.

Fact star is equal to i which of course, means that fact is multiplied by the value of i note that you do not need to increment the value of i here in like the in the previous loop simply because that is taken care of by the evaluation of the expression e 3 here.

There is another way of using the for loop computing the factorial n this is more similar to the while loop that we wrote earlier. So, in this case just for illustration as set we initialization expression e 1 as empty. So, actually any of the three e 1 e 2 e 3 may be missing, but the semicolons must still be there. Of course if the expression e 2 is missing; that means, that there is no loop test and the loop will go on forever, but in that case there must be some other way of breaking out of the loop.

We look at one such way later on in the lecture. So, fact assigned one we have done this is initialization we could have done this as a part of expression e 1 here, but just for illustration i have chosen to e 1 empty here And the loop condition is n greater than zero and the after the loop body n minus minus is evaluated. So, this is at rest same as the while loop for factorial computation that we wrote earlier. The third kind of loop which is possible in C is do while loop and this is what the syntax looks like and we have the flowchart on the right hand side of the slide.

(Refer Slide Time: 12:17)



We have the keyword do followed by s statement and then we have while within bracket some expression e which again is the loop test as usual s has to be a single statement.

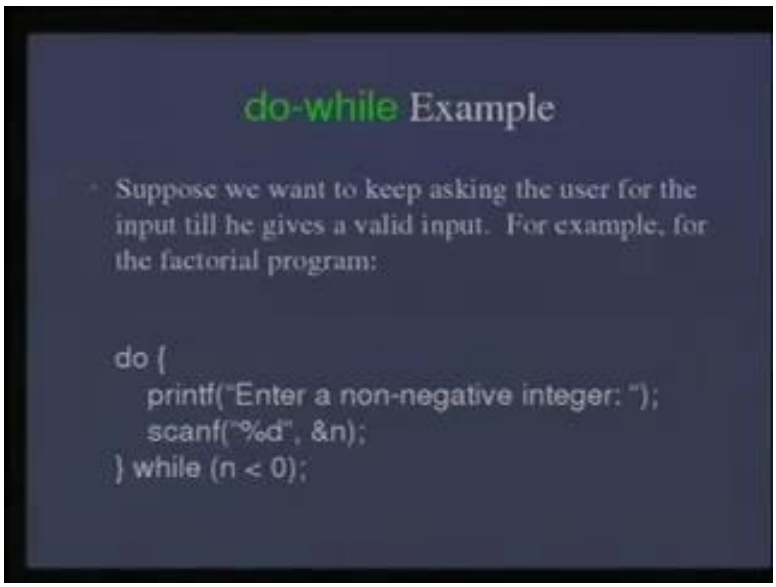
But if you really want to put multiple statement, we can put them together in places and form a single compound statement after that as usual.

So, the difference between this loop and the while loop is that the loop test that is the expression e is evaluated after the execution of the loop body s rather than before. So, in terms of the flowchart this is what it looks like you first execute the statement s and then the expression e is evaluated if it turns out to be false then we break out of the loop because we are going to execute s as long as e is true. So, if e turns out to be true then we go back to the beginning of the loop and execute s once again and then again test e.

So, notice that regardless of the expression e is and whether it turns out to be true or false or whatever s is going to be executed at least which was not the case in the case which was not the case with while loop, because there if the first time the loop test evaluate to false the loop body does not execute even once. So, this kind of loop also has its advantages and we prefer to you do while loop essentially where where we need to execute the body of the loop at least once

Here is a simple example suppose we are writing a program in which we are extracting or reading some input from the user and you want to check for robustness that input is indeed valid

(Refer Slide Time: 13:53)



.

For example in the factorial program we might want to test whether the input number which the user has given is non negative or not if it is a negative integer then the factorial is not defined and. So, we want to ask for another value. So, you want to keep prompting the user for the input till he supplies a valid input and since we have to prompt user at least once we must use we should use a do while loop instead of using a while loop.

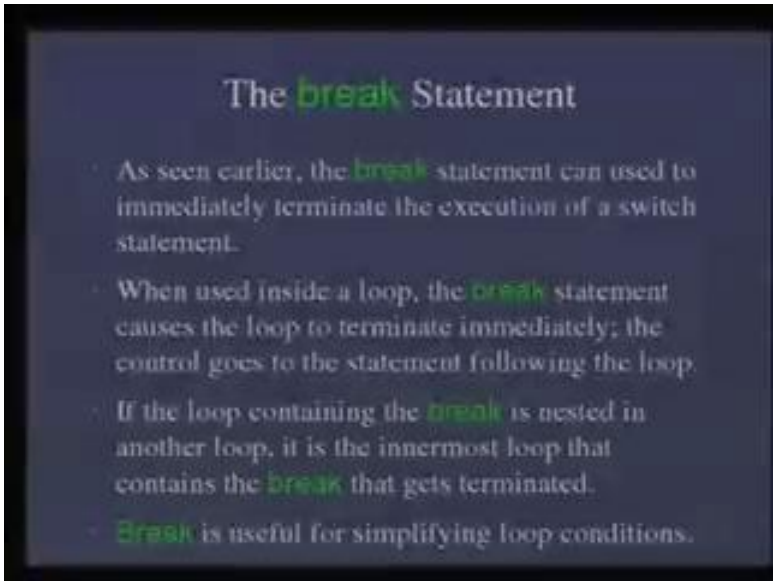
So, here as you can see this is the loop body these two statements and since two statements are not permitted as a loop body we have enclosed them in these braces. So, that this entire statement from this brace to this brace is a single compound statement which can be the loop body

And. So, this loop body prints a prompt for the user to enter nonneg nonnegative integer and then reads the value of the integer that has been in supplied with user into the variable n. And loop condition is n less than zero. So, we want to continue doing this prompting and reading till the user supplies a non negative integer.

So, as long as the suppl supplied integer n is negative we have to keep doing this note that again and the do while loop can also be all do while loops can also be implemented using only the while loop. But it might become cumbersome. So, as an exercise you can try to implement this functionality using a while loop.

You will find that it can be done, but not. So, elegantly. So, these are the three kind of loop statement that we have in C. We now look at some other statements which are useful in the context of loops the first is the break statement.

(Refer Slide Time: 15:52)

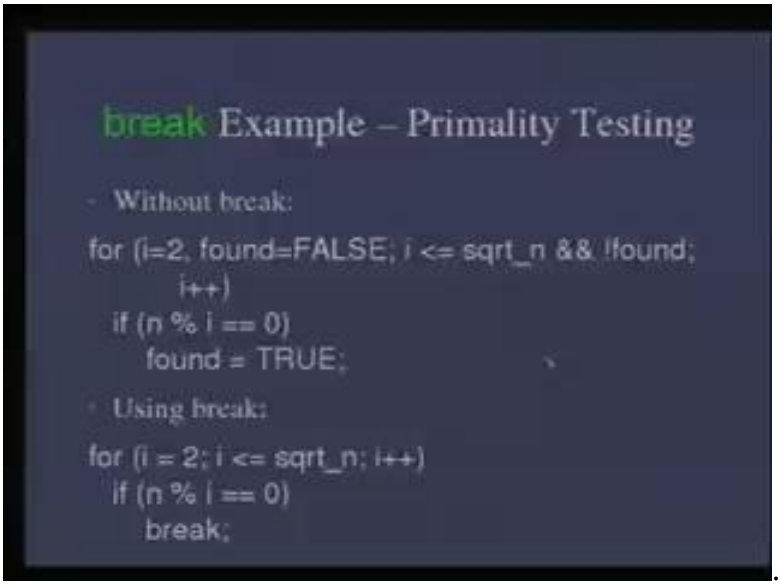


So, break is a very simple statement we have already seen it in the context of switch statement earlier in the last lecture. So, in the switch statement if you recall the break statement causes the execution of the switch statement to finish and go to the next statement that is within the switch statement whenever the break is executed the execution of switch finishes then and there.

So, similarly when used inside a loop the break statement causes a loop to terminate immediately regardless of whether the loop test condition has become true or false and the control goes to the statement immediately following the loop.

Now, of course, it is possible that loop within another loop and within that there is a break statement. So, whichever is the inner most loop that contain breaks statement that is the one which will be terminated by the execution of this break statement. And the reason we use break statement is that it is often useful for simplifying loop condition. So, let us look at an example which we have seen earlier.

(Refer Slide Time: 17:00)



This is an example of finding whether a given number n is prime or not. So, you recall that the way we tested that is by a trying out all divi all possible divisors of n from two up to floor of square root n. And if we find the divisors then of course, n is not a prime and if we do not find a divisor then n is prime and the loop that we had written earlier in one of the initial lectures was similar to this particular loop. Although I have change this slightly and re written it as a for loop instead of writing it as a while loop.

So, initialization is I assigned two and found assigned false false member is a Boolean variable which keeps track of whether we have found a divisor or found a divisor of n yet or not. So, initially found is false, because to begin with we are not found with any divisors and we start looking for divisors from two onwards. The loop condition is i less than equal to square root n. So, i am assuming that this variable square root n is an integer variable which contains the floor of the square root value of n.

So, we have to continue the loop as long as i is less than equal to square root n and not found meaning that if either of these two conditions become false we break out of the loop we stop the loop execution, because if i becomes greater than square root of n; that means, you have tested all divisors up to square root of n and did not find any divisor of n.

And if found becomes true; that means, we have found some divisors and therefore, you must stop the loop execution. And therefore, we must keep executing the loop as long as i is less than equal to square root of n and we have not found a divisor that is we have not

exhausted all the divisors all the potential divisors of n and we have not yet found an actual divisor of n.

And every time after the execution of the loop body we increment i because in the next iteration of the loop we want to try the next high higher potential divisors and in the loop body we just test that if i divides n that is n percent i, which recalls gives the reminder when n is divided by i.

If n percent i is equal to zero; that means, i is a divisor of n and. So, we have found a divisor and. So, we set found equal to true and the next time loop the condition will be evaluated again we find found is equal to true and. So, not found is false.

And we have put this entire condition is also false and. So, we break out of the loop note that this entire loop has become quite complicated because of the use of this Boolean variable found and fact that the loop test consist of two independent conditions both of which needs to be tested. Instead of that suppose we use a break to break out of the loop when we find a divisor. So, in this version of the same loop the loop body is quite simple and the loop header which contains the expression e 1 e 2 e 3 are also quite simple.

The initialization is simply i assigned two as before we have done away with the Boolean variable found we have not using it any longer because we will break out of the loop when you find a divisor. So, there is no need to use a Boolean variable. And the loop test is simply i less than equal to square root n and of course, at the end of the loop body we have to increment i. So, in the we are no longer testing whether we have already found a divisor.

So that means, that if we found find a divisor the loop will not stop and keep executing because i may still be less than or equal to square root n. Therefore when we find a divisor we have to use some other way of stopping the loop and that we do using the break statement. So, if n percent i is equal to zero we execute a break statement and this break statement will immediately terminate this loop. Because this is the inner most loop containing the break statement note that if is not a loop and. So, this break will not do anything to the execution of…

If the for which is the loop which is the inner most loop that contains break. So, as soon as break is executed this for loop gets terminated and the control goes to the following instruction.

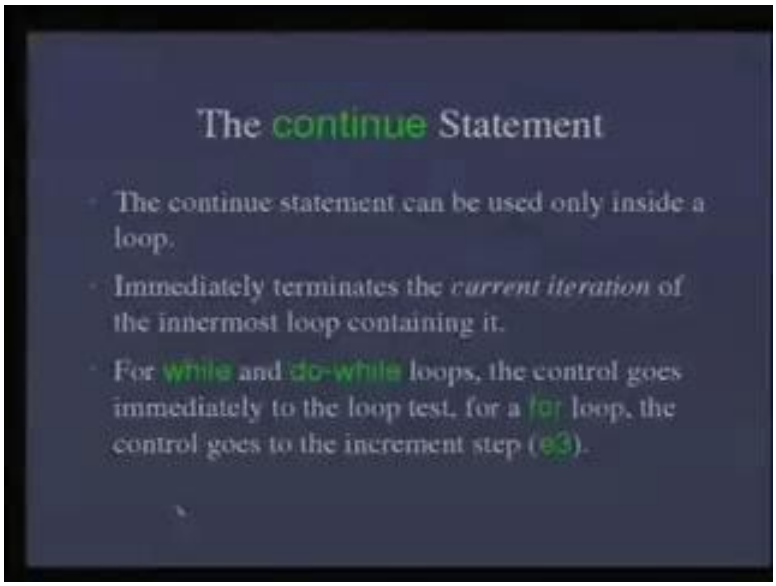
So, of course, after the end of the loop we need to find out whether we have found a divisor or not in the earlier case it was easy because we had this Boolean variable found and at the end of the loop if it was true; that means, we have a found a divisor other wise we have not found a divisor.

Here we do not have this Boolean variable, but finding out whether these came out of the loop because of this break or because of this loop test is quite easy if we check the value of i and if you find that it has become greater than square root n; that means, we must have come out of the loop because of this condition. Otherwise we must have come out of the loop because of this break in which case we found an actual divisor of n and. So, therefore, n is not a prime, related to the break statement there is another statement which is called the continue statement and the the functionality of the continue statement is that within a loop whenever a continue statement is executed the current iteration of loop is terminated it stops and the next iteration begins immediately

that is within the loop body if the continue statement is executed then even if there is there are more statements in the loop body those will not get executed, and the next iteration of the loop will start again. When we say the next iteration of the loop will start again what we really mean is that in the case of do while loop, and in case of while loop the control will go back will will go to the place where the loop condition is evaluated and is tested whether it is true or false.

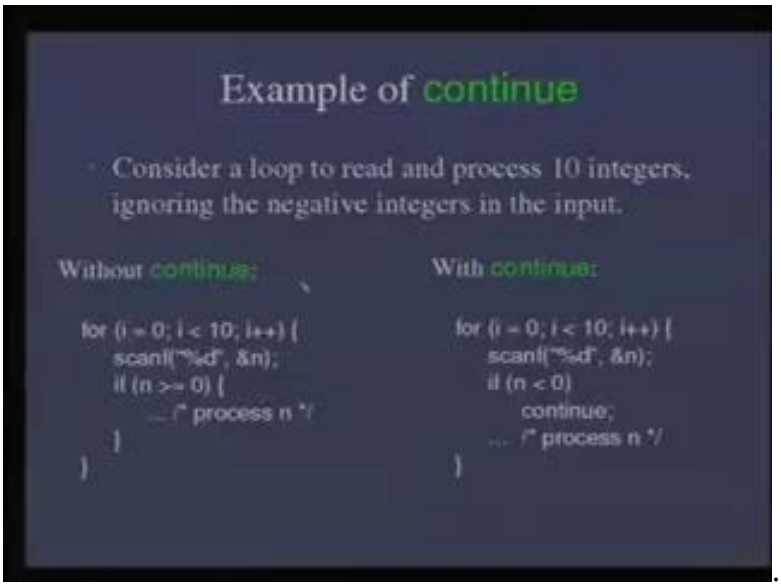
But in the case of for loop some thing slightly different happens that when the continue statement is executed the control goes back to the increment step that is the e 3, where which is the step which is the expression evaluated after the loop body is evaluated.

(Refer Slide Time: 23:17)



So, essentially the way continue statement should be thought about is that on execution of the continue statement the loop body gets terminated, and then but the loop itself does not get terminated. So, in the case of the for loop the expression e 3 is evaluated again and then the control goes back to the loop test and in the case of other two loop other two kinds of loops the control goes back directly to the loop test. And the reason we need to use the continue statement is that it can help in simplifying the loops. For example, again here is a simple situation that you might encounter in some programs that you might write.

(Refer Slide Time: 24:00)



Suppose you want to write a loop which reads ten integer from the input and processes them one by one in certain fashion. Now, this processing could be very simple or it could be extremely complex and the constant is that if there is a negative integer in the input that should not be processed that should be ignored.

So, suppose that not using a continue statement for implementing this then what'll you need to do is this loop is simple for i is equal to zero i less than ten i plus plus. So, do this loop body ten times read the value of n and if n is greater than equal to zero then process n. Now, the problem with this kind of this kind of structure is that if the processing on n is very complicated and it has several nested control statements one inside the other then indentation level will become very deep.

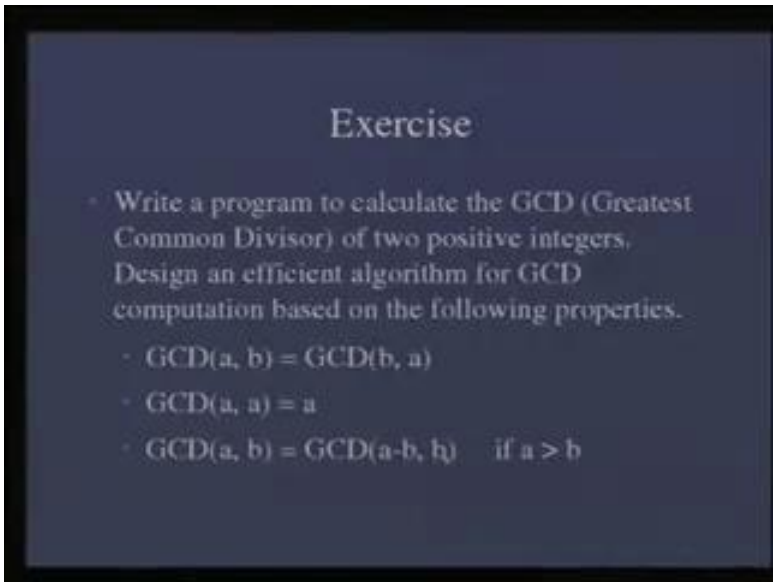
And we are adding to that indentation level by having two indent this entire processing by one extra level, because this entire processing is within the body of the if statement. On the other hand suppose if we use continue statement then what we can do is after reading the value of n in the loop body if n is less than zero, then we put a continue statement; that means, whatever is next in the body of the loop does not get executed and we straight away go to the next iteration of the loop.

Remember that for the for loop when we go to the next iteration that is when we execute the continue statement the control will first go and evaluate the expression e 3 which in this case will increment i and then the loop test will be executed again which is of

course, you want which is of course, what we wanted to do, because we want to read at at least ten numbers regardless of whether they are positive or not.

And so if n is less than zero we continue and otherwise we have this code for processing n note that the control will come to this place only if the value of n is greater than equal to zero. And. So, therefore, we do not need to test whether the value of n is greater than or equal to zero it is guaranteed to be equal to zero the difference between these two implementations of the same functionality is that the processing of n is as one lesser level of indentation when we have used continue. Because this processing of n is not within any if it is directly within the for statement at the next level.

(Refer Slide Time: 26:32)



So, at the end of this lecture here is an exercise for you and this exercise requires you to write a program to compute what is known as greatest common divisor or the gcd of two positive integers you probably are aware of the definition of greatest common divisor of two integers. The gcd of two integers is the largest integer which divides both of them historically the the best known algorithm for gcd is attributed to Euclid who came up with a very efficient algorithm for gcd based on the following properties. So, I will list these properties explain this properties and leave it for you to design the algorithm and then implement that as a program.

So, these are the three properties the first property is GCD of a and b is of course, is the GCD of b and a, that is the arguments can be interchanged. The GCD of a and a is the

same as a that is if the two numbers are equal then the gcd is equal to the same number that is of course, because a is the largest integer that divides a, so that extremely true.

The third property is most interesting if you are given a and b two numbers a and b and lets assume that a happens to be larger than b, then the GCD of a and b is the same as GCD of a minus b and b.

That is if you subtract smaller number from the larger number and GCD of this number resulting number and the smaller number is the same as the GCD of the original two numbers.

So, you can try and convince yourself that this property in deed holds always based on these three properties you should be able to come up with an efficient algorithm for computing the GCD. It is a nice algorithm for computing GCD, we will try to find the largest integer of a which also is a divisor of b, but that is inefficient this will this this properties can lead to an Euclid to an efficient algorithm which will compute the same result in much lesser number of iteration. So, that left us an exercise today in the next lecture we will talk about arrays which are very, very important tool in again writing a nontrivial programs, and allow us to handle large volumes of data rather than small number of data items are designed.