Introduction to Programming in C
Department of Computer Science and Engineering

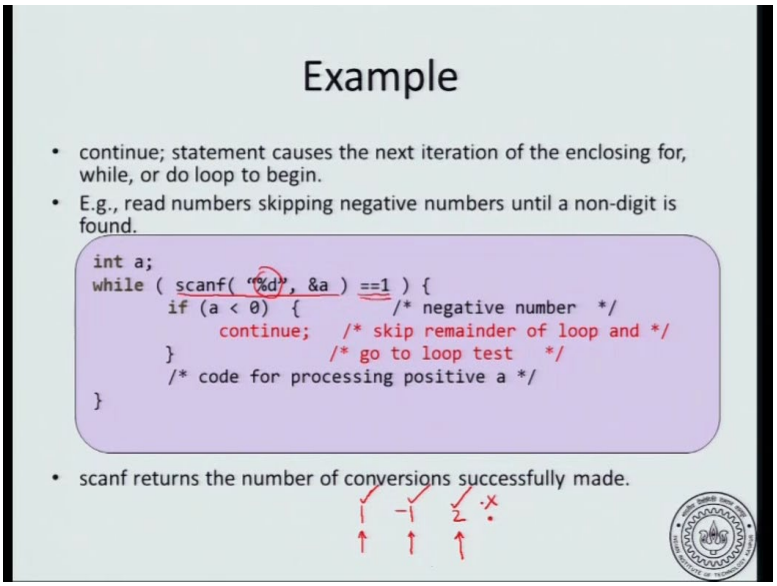
We have seen the break statement, which is a statement used when you are in the middle of a loop and you encounter a condition and you want to exit the inner most loop. There will also be occasions in a program, when you are in the middle of a loop and you encounter some condition and then, you realize that you do not need to execute this iteration, you just can go to the next iteration. So, skip the current iteration.

The break statement was, you encounter a condition and you say, I am done, I will exit out of the inner most loop. Here, it is not exiting out of the inner most loop, it just skipping the current iteration.

(Refer Slide Time: 00:45)

For this, we will see the continuous statement and let us motivate this by an example.

(Refer Slide Time: 00:48)



So, continue the statement causes the next iteration of the closest enclosing for while or do while loop. Let us motivate it with a very simple example. Let us say that we are reading numbers coming in a stream and what we have to do is to skip the negative numbers. So, we have to read all the positive numbers and reading should be finally over, when you encounter some input which is not a number. How do we do this?

Let us imagine that you have the main and things like that written and the central part of the code can be analyzed as follows. So, you have integer variable a and let us examine the code in closer details. So, what we need to do is, we may have an input sequence that looks like this, 1, -1, 2 and then .. So, let us say that the input sequence is something like this. What we will do is, we will do this scanf operation to get the numbers.

So, scanf operation will read the first entry as 1, it will read the second entry as 1. The third entry as -1 and the third entry is 2 and so, on. So, that is what is scanf("%d", &a), we are already familiar with this. But, what does = 1 mean? So, this is something that we have not uncounted. So, far the scanf statement has a return value, it gives you the number of inputs that was successfully read.

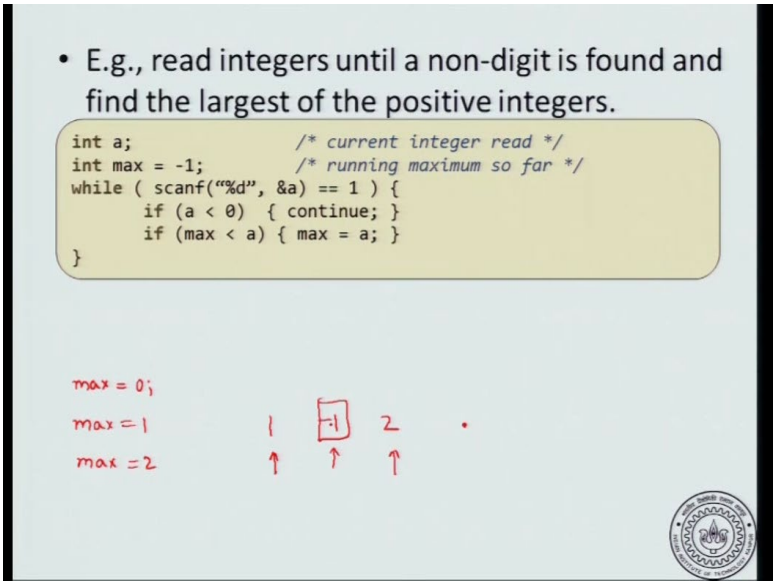
For example, we are trying to read an integer in the %d specifier. So, when we try to read the first entry, it should succeed. So, this will succeed, when you try to read the second entry, it should succeed, when you try to read the third entry, it should succeed. In all these, the scanf %d will return a 1. Because, one entry has been read correctly. Here, it will fail, because it tries to read a natural number here, but what it see is a ., a full stop

character and that is not a number.

So, scanf %d will simply fail. So, this is what I said, it returns the number of conversions that have been successfully made. So, when you try to read an input like 1, -1, 2, . it was succeed in a first three scanf and the last scanf, it will fail. So, that is what the scanf is supposed to do. So, as long as you have read a number.

So, while you have read a number, you examine whether it is a positive number. If it is a negative number that is, if a < 0, then you say continue which is saying that, I do not need to execute the remaining part of the loop. So, this part of the loop will be skipped, if a < 0. Continue means, go from here and start executing the next iteration of the loop. Let us go head and complete the code.

(Refer Slide Time: 04:31)



So, let us modify the problem as a little bit, read the integers until a non digit is found. And let us do something with the positive integers. Let us say that we have to find the largest of the positive integers. So, what should we do? Again, let us try to do it by hand to get a feel for, what I will be doing? So, I have 1 -1 2 .. Let us say that I initialize the maximum to some reasonable value. Since, we are looking at the largest of the positive integers I can initialize maximum to 0.

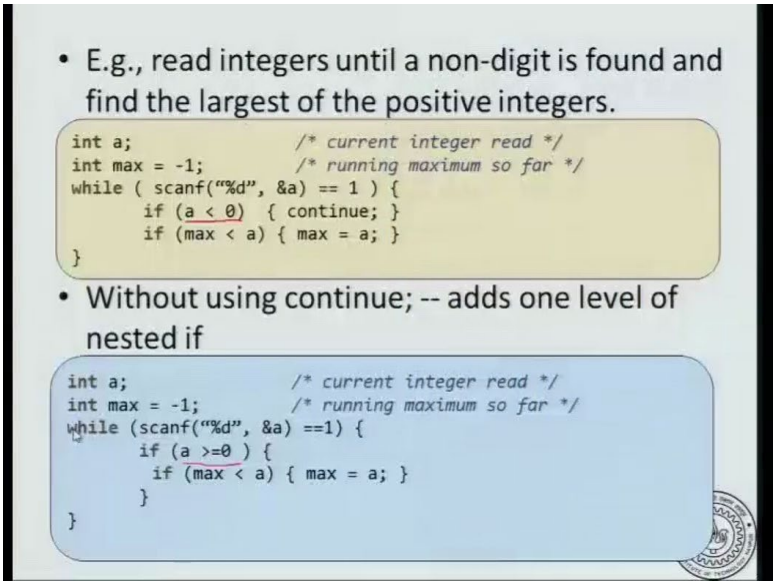
Then, I look at the first one, the maximum read. So, for. So, it is a positive entry,. So, I will update max = 1. Then, I read the next number and it is a negative numbers,. So, skip it. Then, I read the third number which is a positive number. So, I will update the maximum to 2. So, this is the part that we want to focus, if it is a negative number, skip.

So, here is the code for doing that, while the currently read input is a number, that is why the %d succeeded and one entry was correctly read.

So, if the number was read, check whether the number is negative. If the number is negative, continue. Continue means go to the next iteration of the loop. Do not do, what is remaining in the loop. So, if the currently read number is non negative, what you will check whether their current maximum is less than the new number. If it is less than the new number, you reset the maximum to the new number.

So, this is the code that we have written similar to other codes that we have seen. So, you update the maximum number and go and read the next number. If the currently read number is negative, then we will say continue. So, we will not update the maximum. This is what the continue is supposed to be.

(Refer Slide Time: 06:58)

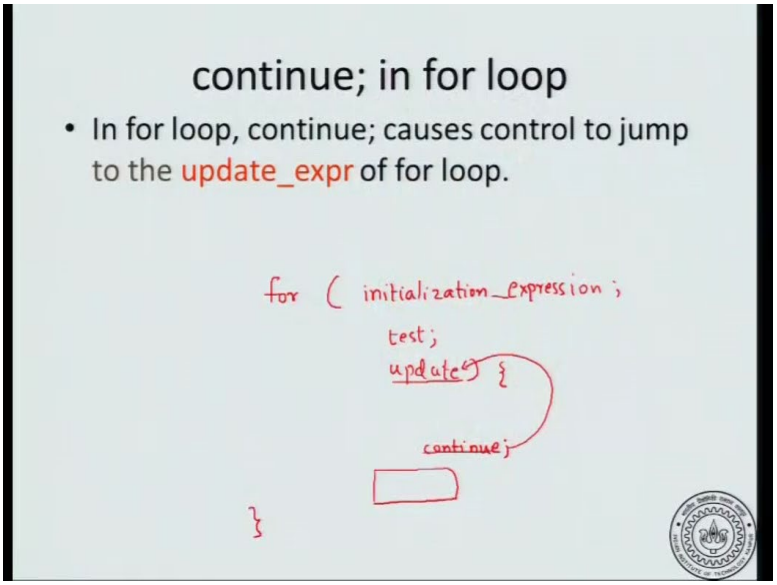


Now, as in break you can also write equivalent code without using the continuous statement. So, let us try to do that and for doing that all we have do is, make sure that the maximum is updated only if it is a non-negative number. So, this says if it is a negative number, do not do the next statement. This says, if it is a non-negative number, then update maximum if necessary. So, it can be written with one more level of nested if. So, this is that if a is non-negative, then execute the next statement. Here. it says that if a is negative, then continue which means skip to the next statement.

So, notice that these two conditions are the negations of each other. The long and short of it is that continue is not really necessary. But, if you have it, then it is useful and it makes

the code clearer in certain occasions.

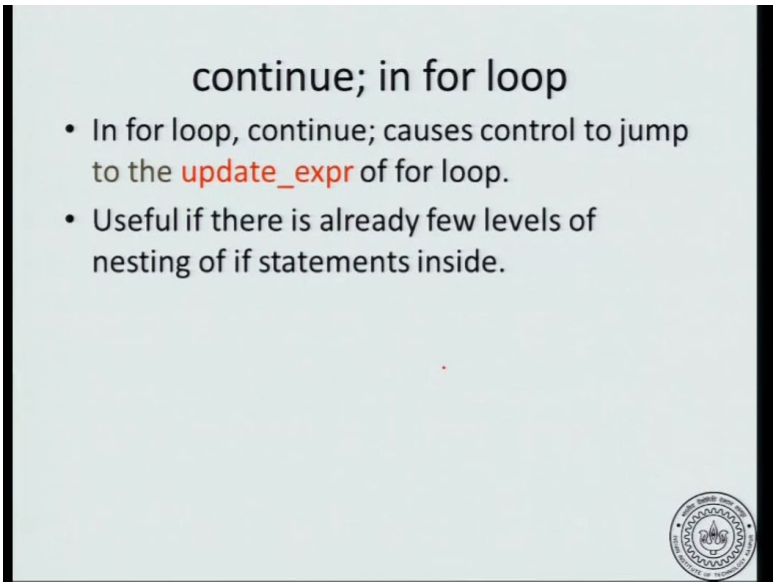
(Refer Slide Time: 08:10)



What happens to continue in a for loop? Noticed that, for loop has the following form, you have for, then there is an initialization expression. Then, there was a test and finally, there was update and then, you have the body of the loop. What happens if you encounter a continue in the middle of the loop? In the case of a while loop, it is very clear, you go to the test expression, you go to the next iteration.

The only contention is, in the case of a for loop, do you go to the update statement? And the answer is yes, then you skip the remaining part of the loop. So, this is the remaining part of the loop that you would skipped. When you skip that you go directly to the update statement. Notice that, when you do the break. So, if the statement has a break, you break immediately out of the loop without doing the update. In the case of a continue, you have to do the update.

(Refer Slide Time: 09:37)



And as with the break statement, the continue statement is also redundant, you can program without using the continue statement as well. But, it is useful if whereas, already a few levels of nesting of the if statements inside it. We saw in the previous slide that, you could avoid continue statement by using an extra level of nested if statement. Now, if you do not want to complicate the code in that way, you can use a continues statements.

Otherwise, in other cases you may want to exit out of the loop, in that case you can use the break statement. So, they are extra feature that the C language provides, they are not really necessary, but they are used with.