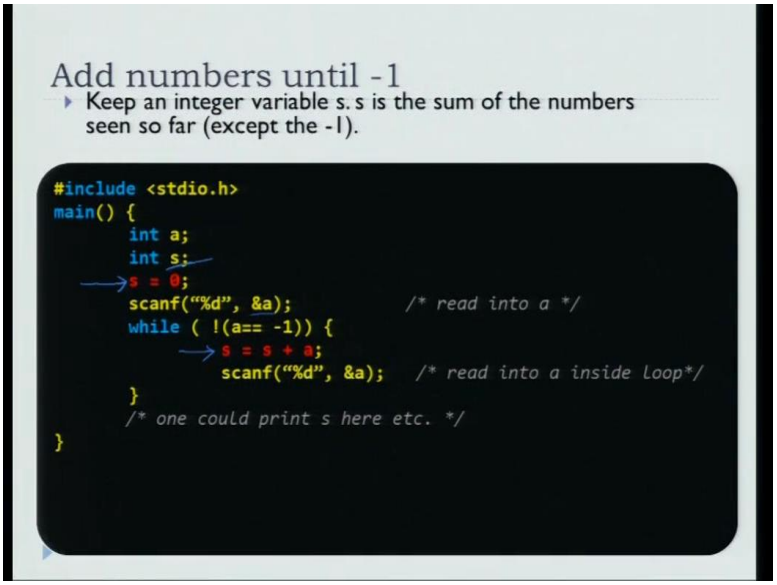
Introduction to Programming in C
Department of Computer Science and Engineering

In this session, we will continue the program that we were writing. Recall that, we were writing a while loop, which will read a bunch of numbers. And it is supposed to sum them up, until you hit a -1. In the loop that we have seen, so far we just read the numbers until -1 was encountered. So, let us now complete the program and compute their sum as well. So, for computing there sum, how do we normally do it?

We will add numbers two at a time. So, the first two numbers will be added. Then, that sum will be added to the third number and so on, until you hit a -1. So, let us try to do that, in the course of a while loop.

(Refer Slide Time: 00:46)



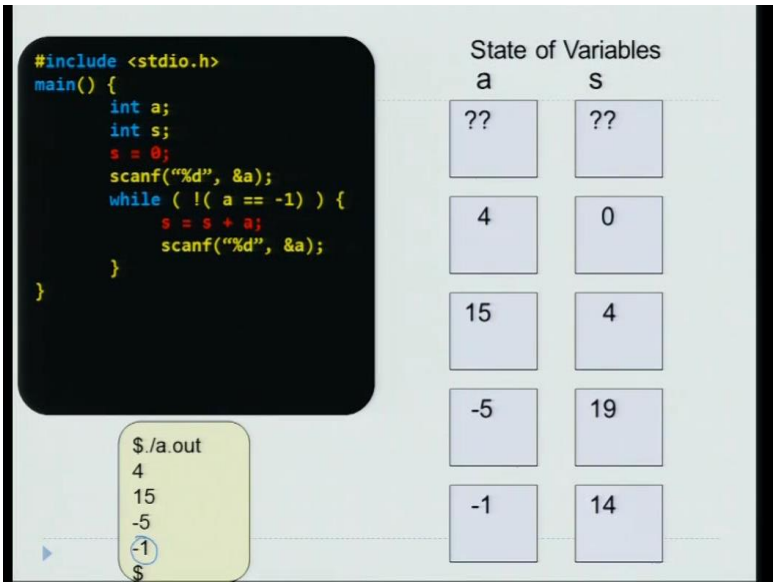
What I will declare is, I will declare a new variable s. So, here is the new variable s that, I have declared. s is supposed to the hold the sum of the variables that, I have read so far. Now, it is very, very important that, when you declare a variable, you should initialize them properly. In the case of a, we did not initialize it because, we were reading the first number, as soon as was declared.

But, in the case of sum, you would use s to maintain the sum, as you read numbers. So, it is important that, you start with s = 0. So, the initialization step marked by this arrow is quite important. If you do not initialize it properly, then the sum may not be correct, as

we will see soon. So, we keep a variable s, which is supposed to hold the sum of n numbers, sum of these numbers and initialize the sum to 0.

Then, the difference from the loop that, we have seen so far is highlighted in red. So, earlier recall that, we were reading the number. And just testing, whether the number is -1, if it was not -1, you read one more number so, that was the loop. Now, inside the loop, what we will do is, we will keep up, running sum of the numbers that we have seen so far. So, initially s sum is initialized to 0. Then, if the first number is not -1, you add the first number to s. So, s will now be the first number. Now, read the second number. If the second number is not -1, you will enter the loop again. So, you will add the second number to s. So, s is now first number plus second number. And this keeps on going until you hit -1, in the input. So, let us continue with this.

(Refer Slide Time: 02:47)



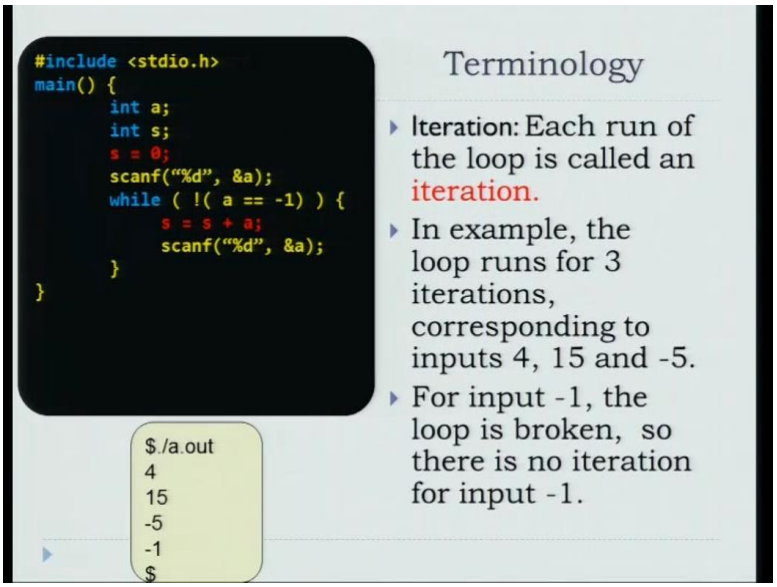
Let us try to trace the execution of this program, on a sample input and try to understand, how it works. Let us say that, I compile the program successfully and run the program. So, I run a dot out and let us as before, let us the first number be 4. So, after initialization, when you declare the variable a is undefined and s is also undefined. After the initial statement, s = 0, s is now 0. And then, you scan the variable a. So, a becomes 4, because 4 was the input. And sum is still 0. You enter the loop and you say, s = s + a. So, sum becomes 0 plus 4, which is 4. And you read the next number. Let us say the next number was 15. So, a becomes 15, a is not -1. Therefore, you enter the loop again. And

sum is now 4 plus 15, which is 19. So, sum at any point of time is the sum of the number that, we have read so far. So, we have read 4 and 15. So, the sum is 19.

Now, you read the next number. Let us say, the next number was -5. -5 is not -1. Therefore, you enter the loop again s equal to s plus -5. So, s becomes 14. Then, you read the next number. And let us say, the next number was -1. So, since the number read is -1. You go back to the loop. And this condition becomes false. So, you exit out of the loop. And then print that, the sum is let us say 14.

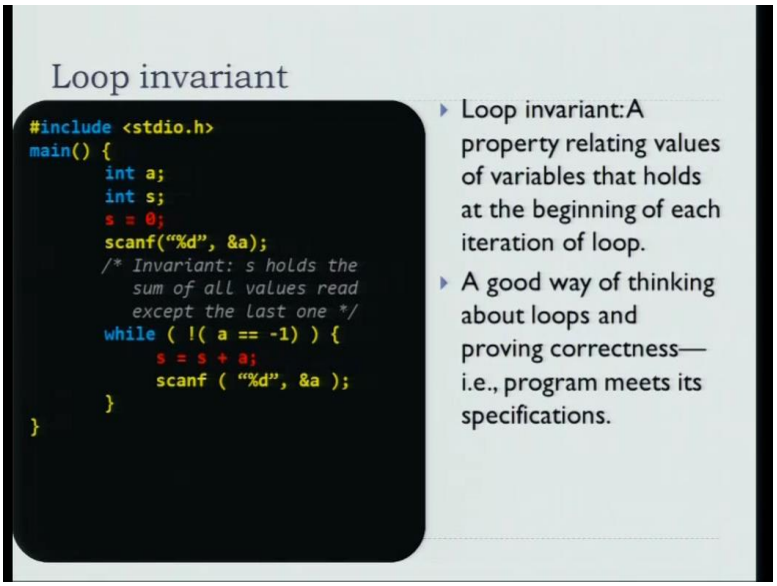
So, when you verify it by hand, you would see 4 plus 15 plus -5 is 14. So, you have, the program has executed correctly. The important thing to note is, the final -1 is not summed up. So, that is, it is used as the end of the input and you should not compute the sum of the numbers, including -1. -1 is excluded. Then, the program executed, correctly.

(Refer Slide Time: 05:42)



We will introduce a few terminology associated with the notion of a loop. Each execution of a loop is known as an iteration. So, in the above loop, when the input was 4, 15 -5 -1, the loop runs for three iterations, corresponding to the inputs 4, 15 and -5. So, for input -1, the loop is broken. So, you do not enter the loop. So, you do not count an iteration corresponding to -1. So, you entered four numbers including the -1 and the loop executed three times. So, you say that, the loop had three iterations. So, this is a technical term associated with the loops.

(Refer Slide Time: 6:16).

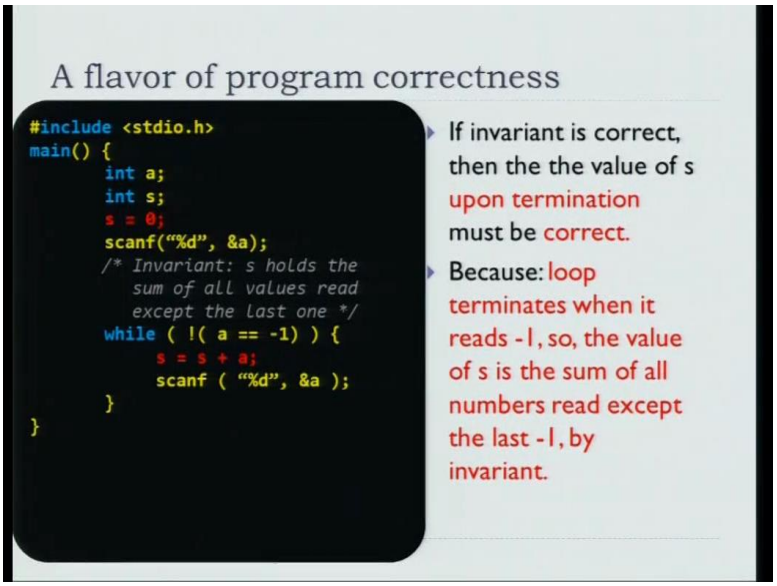


And here is a concept that, I will introduce to help you argue about the correctness of a loop. So, there is a notion known as a loop invariant. Now, a loop invariant is a property relating values of the variable that holds at the beginning of each loop. So, thus bit abstract let me just illustrate with the example, that we just saw. So, loop invariants are a good way of thinking about the correctness of loops that, you have to do.

So, in our program what will be the loop in invariant? Let us look at the property of that, we are interested in. There are two variables in the program, s and a. And both of those variables are involved in the loop. But, the interesting property that we have relates to s. What is the property that, s holds with respect to the loop. So, we can see that, s holds the sum of all values read so far, except the last value is that true, the first time that we enter the loop? Yes, because s was initialized to 0. And you had actually read the number.

So, it is true that, s holds the sum of all values, except the first one. So, that is true, when you first enter the loop. And that, any point when you enter the loop, you sum the last value that was the read. And read one more number. So, you will see that, s still holds the sum of all values read so far, except the last one. So, this is the loop invariant in the program. And loop invariants help you, argue about the correctness of the loops.

(Refer Slide Time: 08:13)



So, if the loop invariant is correct and the program maintains loop invariant, then the value of s when the program stops, will be correct. Why is that? Because, the loop terminates, because the last value read was a -1. And the invariants says that, s holds the sum of all values, except the last value. So, this means that s holds the sum of all numbers, except the -1.

Therefore, when the program ends that is, you exit out of the loop, s holds the sum of all number that you were supposed to add. So, here is how arguing about loop invariant and seeing, whether loop invariant holds in the loop that you have written, helps you argue that the program is correct.