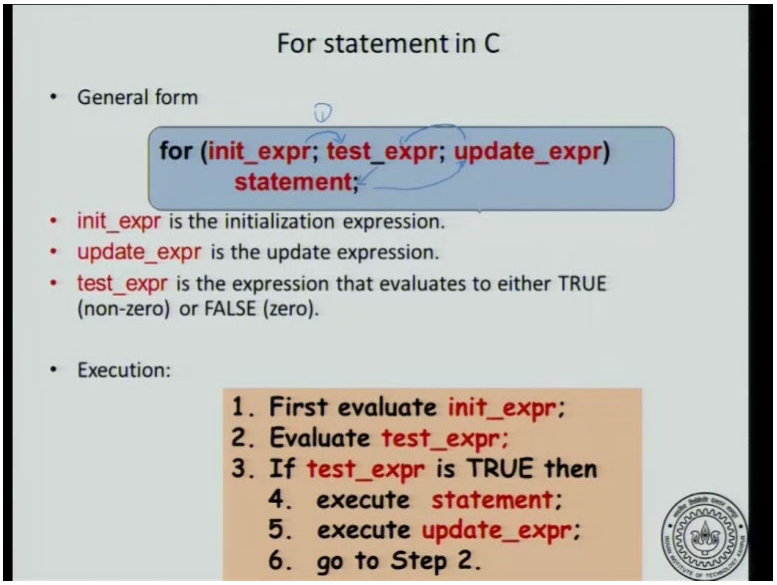
Introduction to Programming in C
Department of Computer Science and Engineering

In this session we will see a very popular loop construct in C.

(Refer Slide Time: 00:07)

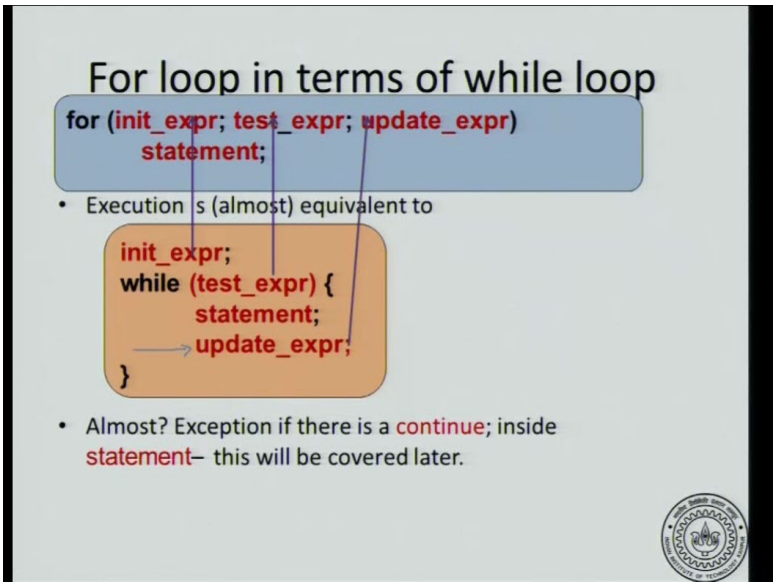


We have already seen while loops and do while loops, will see that do while loops are not all that common in C code, when C programmers code. Among the most popular loop construction C is this, for loop. So, let say what it stands for? The expression for the general form of the for statement, the slightly more complex than that of a while loop. While loop was very simple, while as the certain expression was true, you execute the statement and when the expression becomes false, you exit out of the loop, for loop is slightly more complex.

So, it has the following components, it has an initialization expression, then the test expression, this the expression corresponding to the expression inside the while loop and then there is an update expression, followed by the loop statement. This looks complex at first,, but it is quit intuitive once you start using it. The execution is as follows, first you execute the initialization expression, then you test whether the test expression is true or not.

If the test expression is true, you execute the statement and then come back and execute the update expression. After you execute the update expression go back to step 2, which is go to the test expression. So, init expression is the initialization expression, update expression is the update expression and test expression is the expression, that is evaluates to either true or false.

(Refer Slide Time: 01:54)



So, if you look at the flow of how the code goes, then it is first you start from the initialization expression, then you go to first you start from the initialization expression, then you go to the test expression. If the test expression is true, you go to the statement, then you go to the update expression and you go to the test expression again. So, the loop is here you test the expression, execute the statement, update and test again, initialization is done only once. So, this is the first step and here is the loop, this sounds bit complex at first,, but it is quite simple to use, once you get the hang of it.

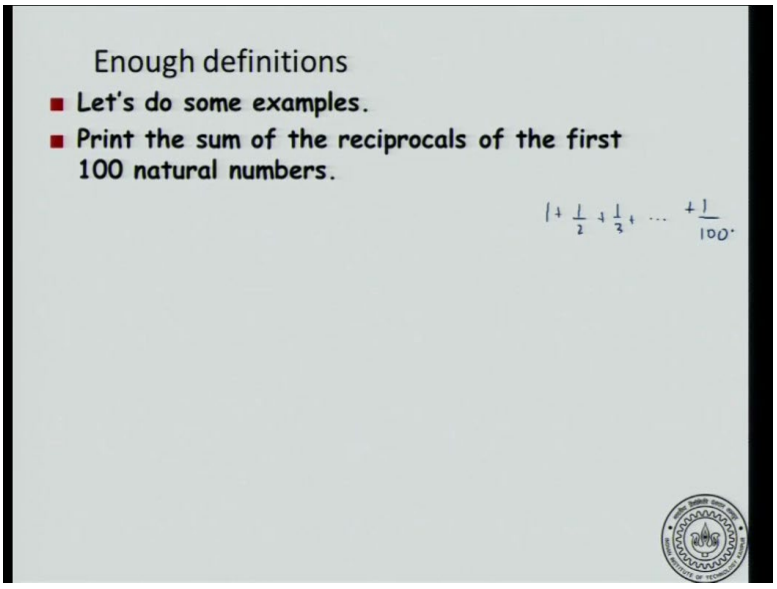
So, the execution of the for loop can be understood in terms of the while loop. The execution of the for loop is almost equivalent to the following while loop, you have the initialization expression before the while loop, then the test expression, while test expression, then you have statement and then you have the update expression. So, if you have a for loop you can write the equivalent code using while loop. So, if you say that I do not want to use for loops, here is how you have a for loop and you can write the equivalent while loop in the following way.

Or if you have a while loop, you can write a equivalent for loop by looking at the this form and how it is translate to the corresponding for loop? Now, there why did I say execution is almost equivalent, we will see this later in the course. Whenever, there is a continues statement or a break statement, you will see that we need to modify this as equivalents between the for loop and the while loop. But, for now for with the features of see that we have seen so far. The for loop is equivalent to the while loop and we will have to modify this slightly later.

So, the init expression maps to the first part of the for loop, the test expression maps to the second part and the update expression maps to the third part. One important thing to notices is that, the update expression is after the statement. So, we have the following first we execute the initialization expression, then we test whether the expression is true. If it is true, you execute the statement, update expression and then again go to the test expression, if it is true you execute statement, update and then test again.

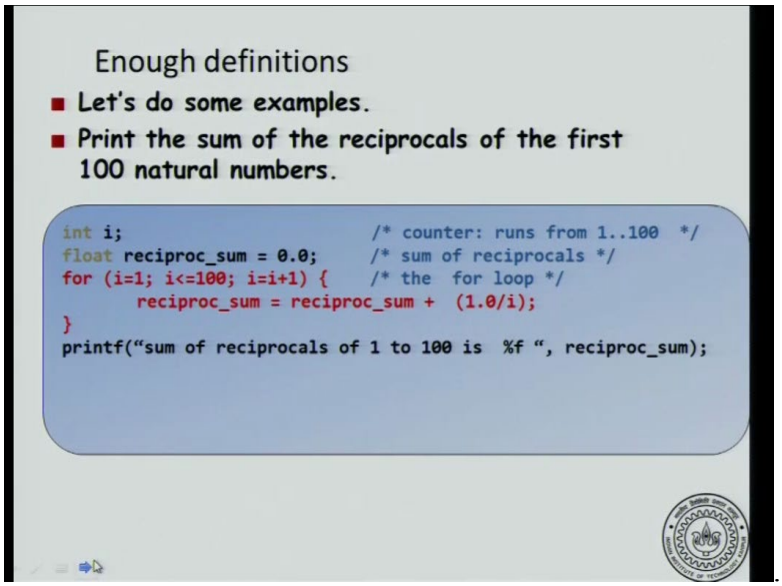
So, you initialize the expression then when the test the test expression if it is true, you execute the statement after the statement is true, after the statement is executed you update the expression and go back to the test expression. Because, that is how you execute it in the while loop? You first initialize, then test whether it is true execute the statement, update and then go back to the test expression. So, this is how a while loop can be translated to a for loop and vice versa.

(Refer Slide Time: 05:23)



So, let us do some examples very simple think, let us say that print the sum of reciprocals of the first 100 natural numbers. So, what do I want to do? I want to do the following, I want to do $1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{100}$. So, how would I do it? I would initialize a variable call sum, sum will be initialized 1 and then 2 sum I will add $\frac{1}{2}$, then to that I will add $\frac{1}{3}$ and keep on going until $\frac{1}{100}$.

(Refer Slide Time: 06:08)

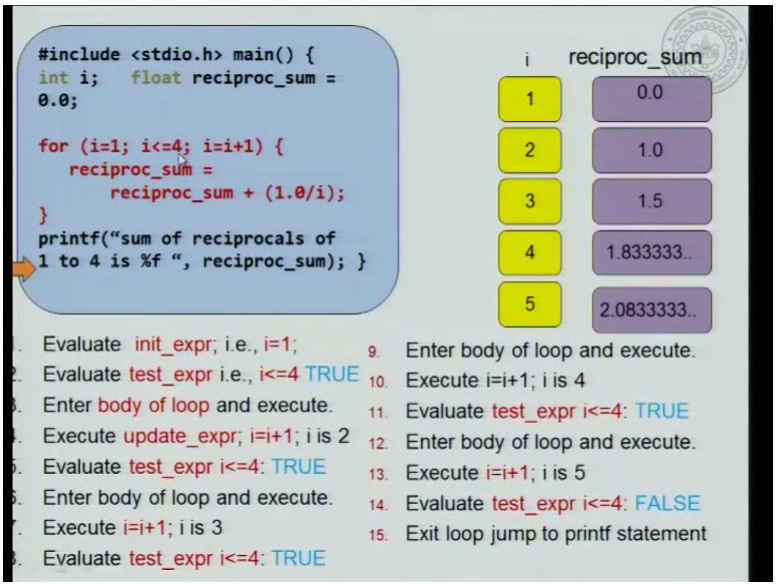


So, let see how to code this in C using the for loop. So, I have a variable call reciprocal sum and even though I am summing over integers, we know that the reciprocal numbers will be real numbers. So, in order to keep the reciprocal sum I need a floating point number, floating point variable and then I have an integer variable, which goes from 1 to 100.

So, here is how I will do the loop? First initialize i to 1, if i <= 100, you enter the loop and do reciprocal sum equal to the current reciprocal sum plus 1 over i. After doing that you update by saying i = i + 1,. So, increment i. Once the increment is done, you test whether the new number is less than or equal to 100, if it is less than or equal to 100, you do the reciprocal sum come back update, until you reach 101.

At the point where you reach 101 you test whether i <= 100 that becomes false and you exit. So, you will see that when you exit out of the loop, the reciprocal sum will be the sum of reciprocals of numbers from 1 to 100. So, here is how the for loop functions.

(Refer Slide Time: 07:45)



So, instead of 100 let us try to executed on a particularly very small number to see how this for loop executes. So, let us instead of summing from 1 to 100, let us sum from 1 to 4. So, first you have the initialization expression. So, i is undefined before you enter the while loop, reciprocal sum is of course initialize to 0. So, I can after initialization i will be 1, as soon as it is initialized we will test whether it is less than or equal to 4, 1 is less than or equal to 4 that is true.

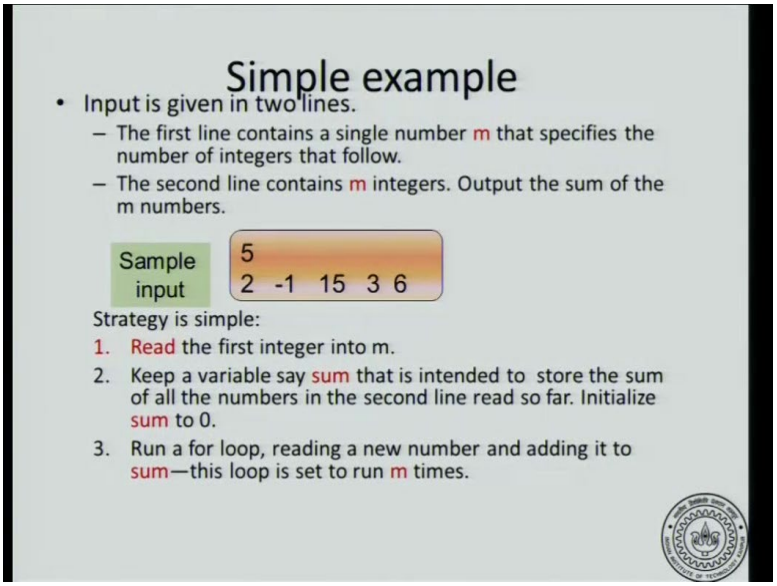
So, you will enter the for loop, then you add to the reciprocal sum 1 over i, i is 1,. So, 1 over 1 is 1. So, reciprocal sum will be updated 2 reciprocal sum plus 1. So, reciprocal sum would be 1, then you go to the update expression, at this point you have i = i + 1, So, i becomes 2. Now, test whether i <= 4 yes it is and enter the loop. So, 1 plus 0.5 then go back to the update expression i becomes 3 now and test whether 3 is less than or equal to 4 it is,. So, enter the loop.

So, you add 1.5 plus $\frac{1}{3}$, 1.833 and. So, on, update again you have 4, 4 is less than or equal to 4 that is true. So, you enter the loop one more time and add 1 over 4.25 to the current number. So, you get 2.0833 and. So, on update again i becomes 5, at this point 5 is not less than or equal to 4,. So, you exit out of the for loop. Now, you say that print that the sum of reciprocals from 1 to 4 is reciprocal sum, which is 2.0833.

So, even though the for loop looks complicated, once you start using it, it is very nice to right, you have a initialization expression, you have a test expression and then you have the update expression, that you should do after every execution of the loop, after every

iteration you should have the update expression. As soon as the update is over, you test whether I can execute the loop one more time, if I can enter the loop update and test again and. So, on, until the loop condition is false.

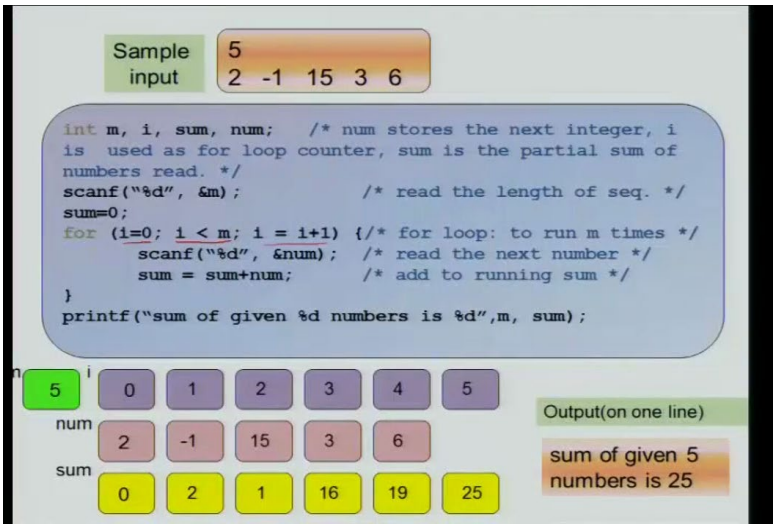
(Refer Slide Time: 11:00)



Let us take another example, you have two lines, the first line contains a single number m, which specifies how many numbers are there in the second line. The second line contains m integers and we have to just output the sum of the m numbers. Now, we know how to do this, we have already done this using a while loop, let us try to do it using a for loop. So, the sample input is let say the first line is 5 and then I have 5 integers on the second line.

The strategy is very simple, you read the number on the first line into m and then have a variable called sum, which will start with the first number and keep on adding the subsequent numbers, until you have read m numbers initialize sum to 0. So, run a for loop from the first number to the mth number and keep adding the numbers to sum. So, this loop will run for m times.

(Refer Slide Time: 12:11)



So, let us code this up, you have m, i, sum and numbers which are all integers. First you scan the number m, initialize sum to 0 this is important. Because, if sum is not properly initialized it is sum garbage value and you keep adding numbers to it, you will get garbage value as the output. So, initialize the number sum properly to 0 and then here is the for loop, what the for loop does is, you start with i = 0 and go on until i less than m.

Now, you could also do the following could start with i equal to 1 and go on until exactly m. So, if you start with i equal to 1 you will say i less than or equal to m, you can adapt either convention, in C it is more popular to start from 0 and go on until m minus 1. So, you break the loop when i is equal to m. So, here is the test condition for the loop and then you have the loop body, which is you read the number and add the number to sum and after you have done that, you have the update expression which is i = i + 1.

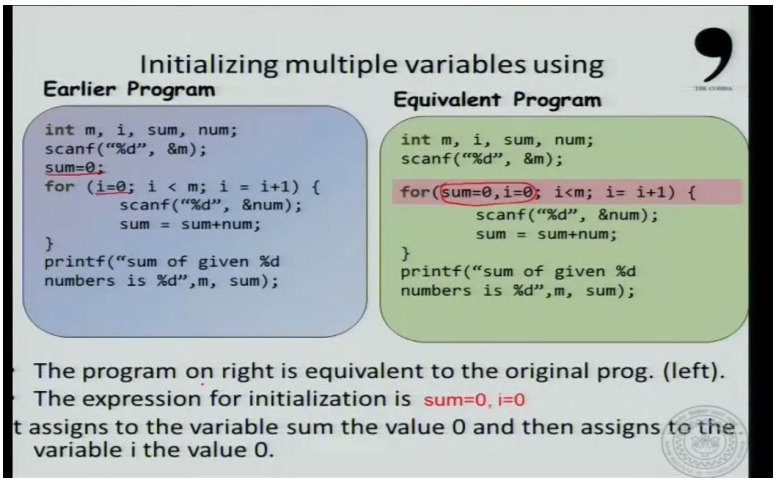
So, here is the how the for loop looks you start from 0 and go on until i becomes m, you add the number and just increment i, which is i is the number of integers we have seen so far. Let us do trace of this execution, you start you have this integer variables and you first read m which is 5, the number on the first line and then we do things in order, you have initialized sum to 0, you start with i = 0.

Once you do the initialization expression i become 0, i is less than m 0 less than 5 that is fine. So, you execute the loop, scan that next number, which is 2 add it to the sum. So, sum becomes 2 now update, update is increment i,. So, i becomes 1 and test whether 1 is less than 5 it is. So, you read the next number add it to the sum,. So, this sum becomes 1 update again and keep repeating this, until you have read all 5 numbers.

So, when you read the 5th number i will be 4, after that you add the 5th number to the summation. Once you done i will be incremented to 5, 5 is not less than 5, 5 is equal to 5. So, you will exit out of the loop, at this point you will have the correct sum,. So, the correct sum will be 25 and you exit on. So, the printf will come out on one line, it will say that the sum of given 5 numbers is 25.

So, what I will recommend is, write the same program using a while loop and a for loop and see how you can easily go from while to for and for to while. The advantage of the for loop and the reason why for loop become,. So, popular among programmers is that, in comparison to the while loop, it is first of all it is easier to read. Because, you have all the initialization expression, the update expression and the test expression all on one line. So, you see what the loop is about. The second is that, it involves fewer lines of code, then the corresponding while loop. So, it is a very popular loop among programmers.

(Refer Slide Time: 16:45)



Now, here is a syntactic convenient that C providers and let me make this remark as the final thing in this session. So, notices that we had to initialize two variables here. So, the first is sum was initialized to 0 and the second was that i was initialize to 0. Now, would not be convenient, if I could do this together and that is what C provides us. So, I have something known as the comma operator. So, the normal comma that we have seen.

So, in order to initialize multiple variables at the same time, I can say sum equal to 0 comma i = 0. So, C will initialize the variables in the order, that it is given, first it in will initialize sum to 0 and then it will initialize the i = 0. So, here is a very synthetically convenient notation that C provide for as the advantage again is that you end up with fewer lines of code.