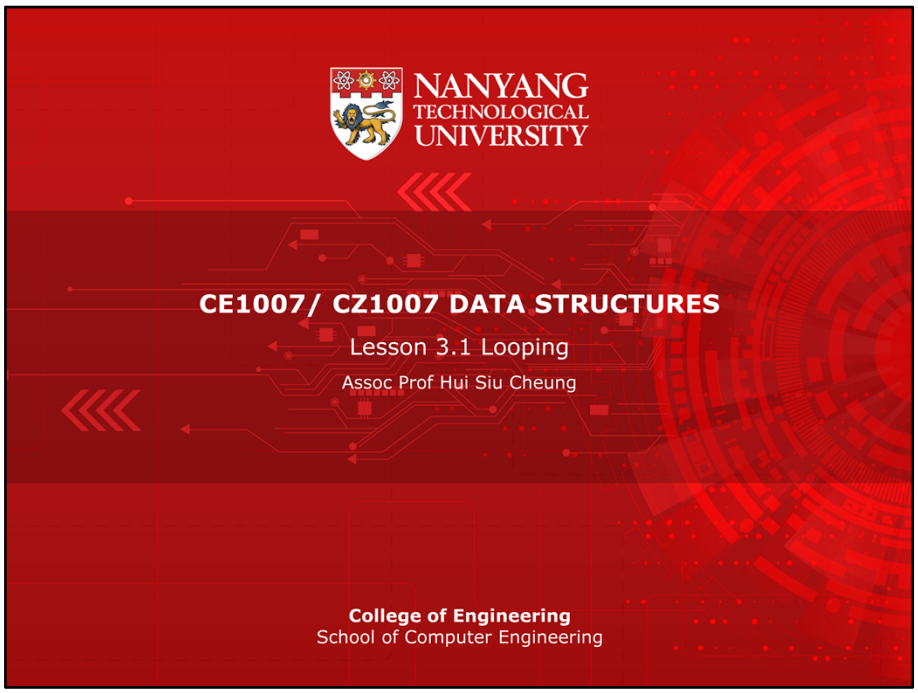
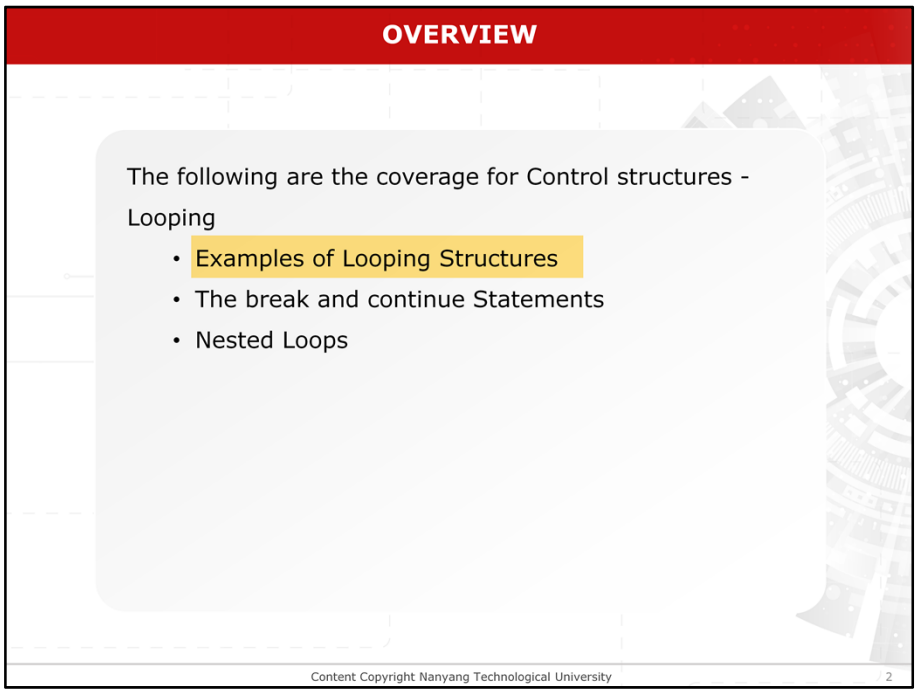
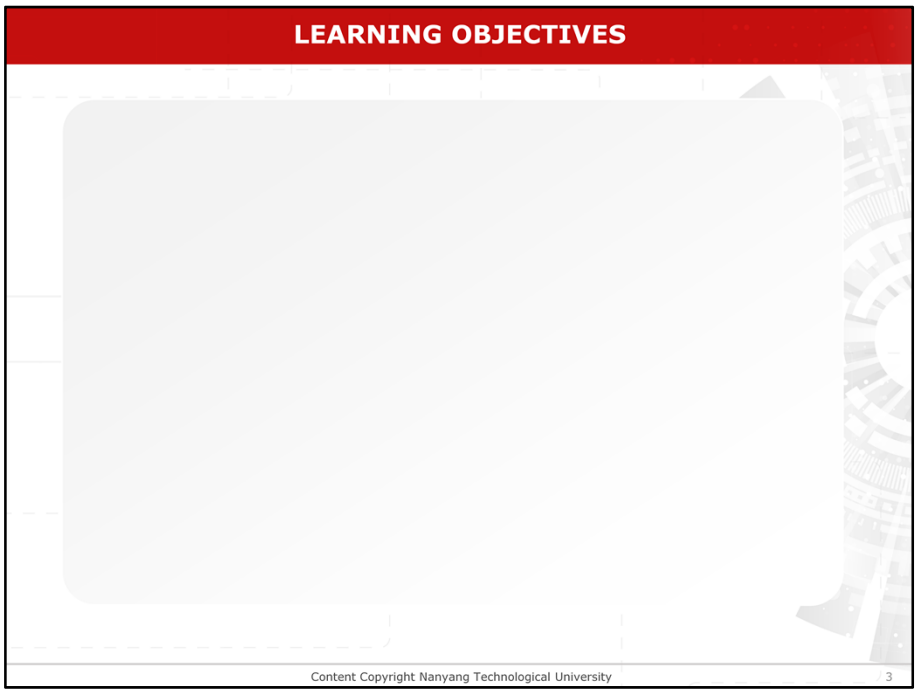


This lesson is on control structures looping

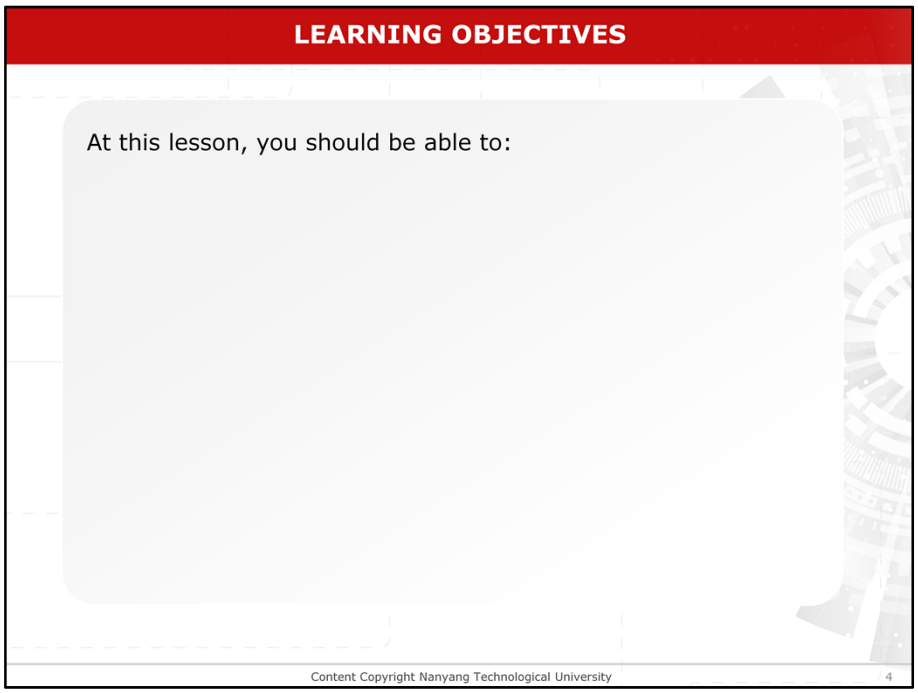


Basic C Programming

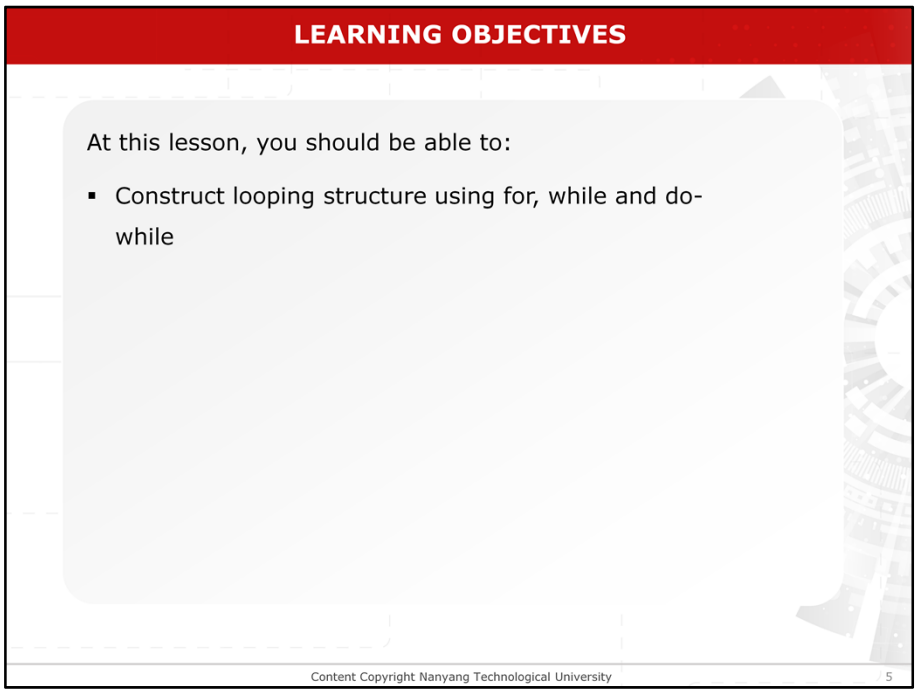
There are 3 main sections to cover for Control structures (looping). This video focused on the 1st topic: Examples of looping structures.



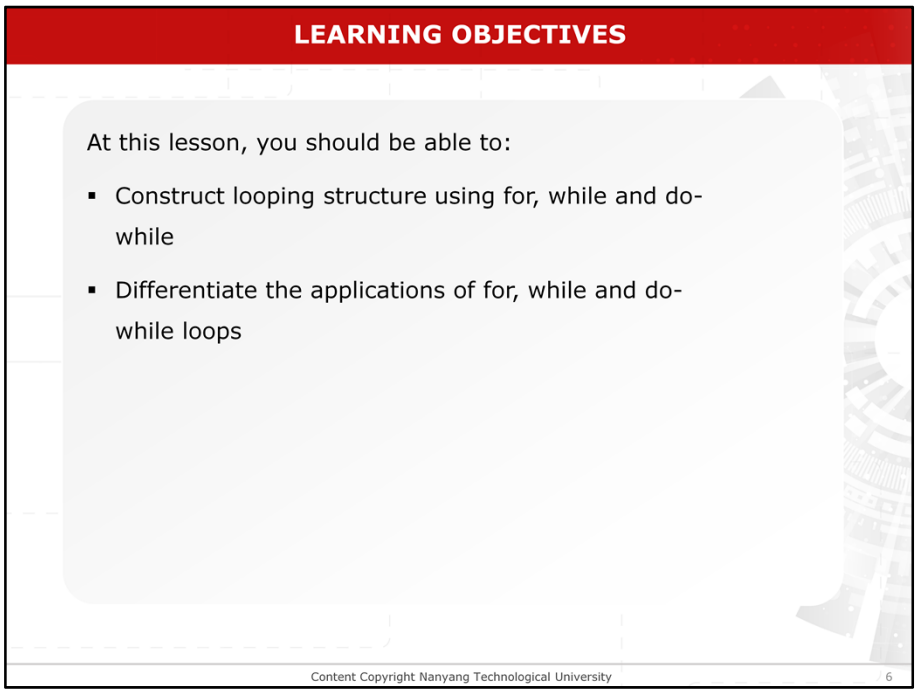
Learrning objectives



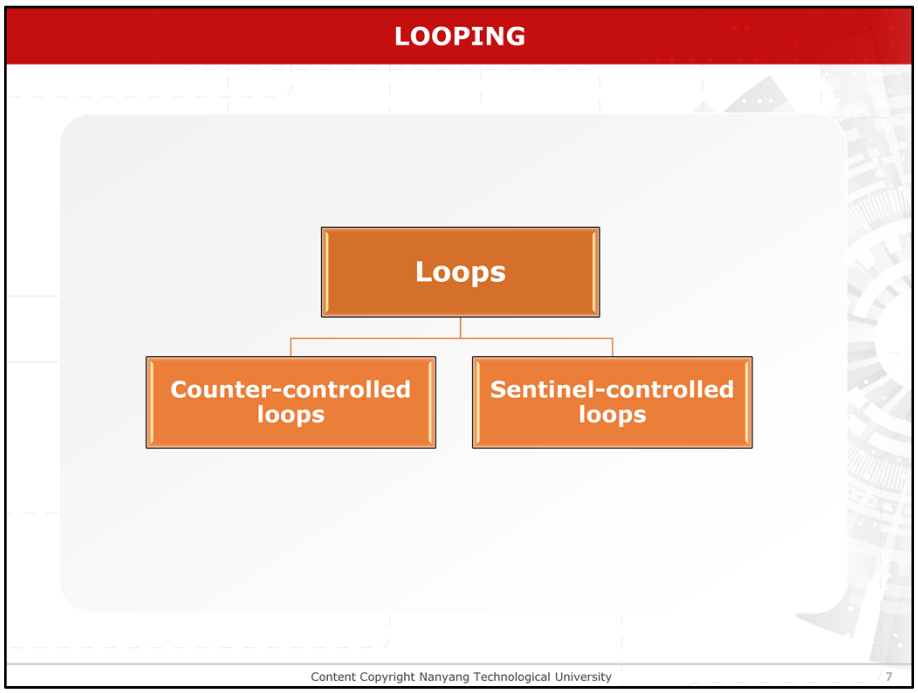
At this lesson, you should be able to:



Construct looping structure using for, while and do-while

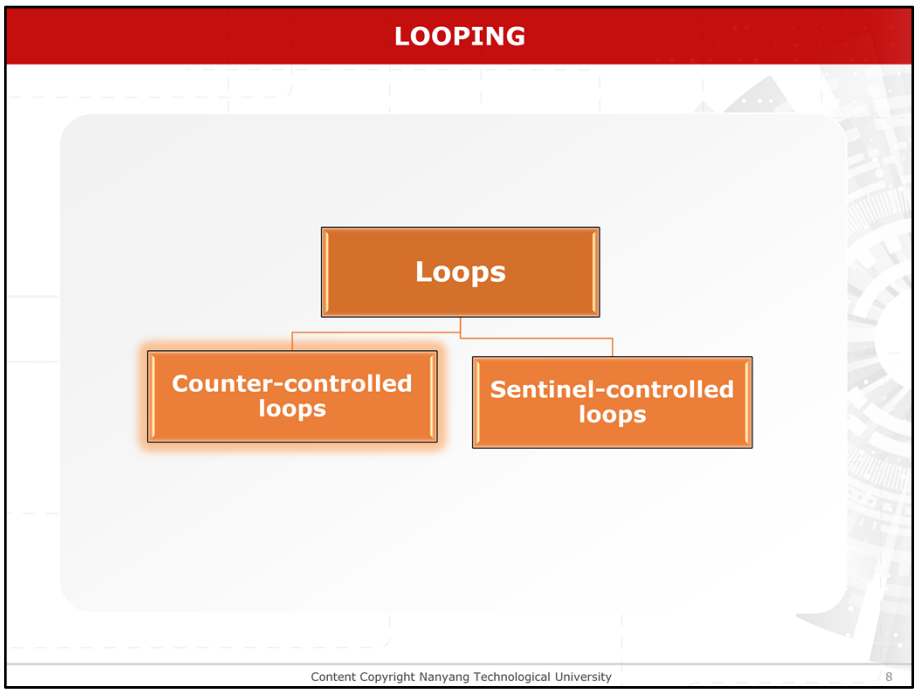


Differentiate the applications of for, while and do-while loops

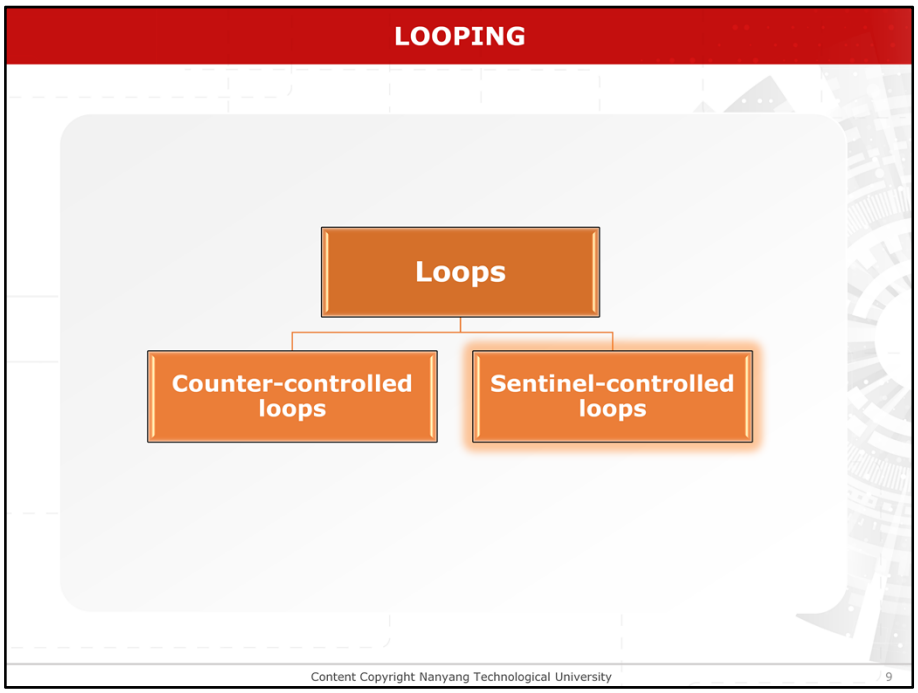


Looping

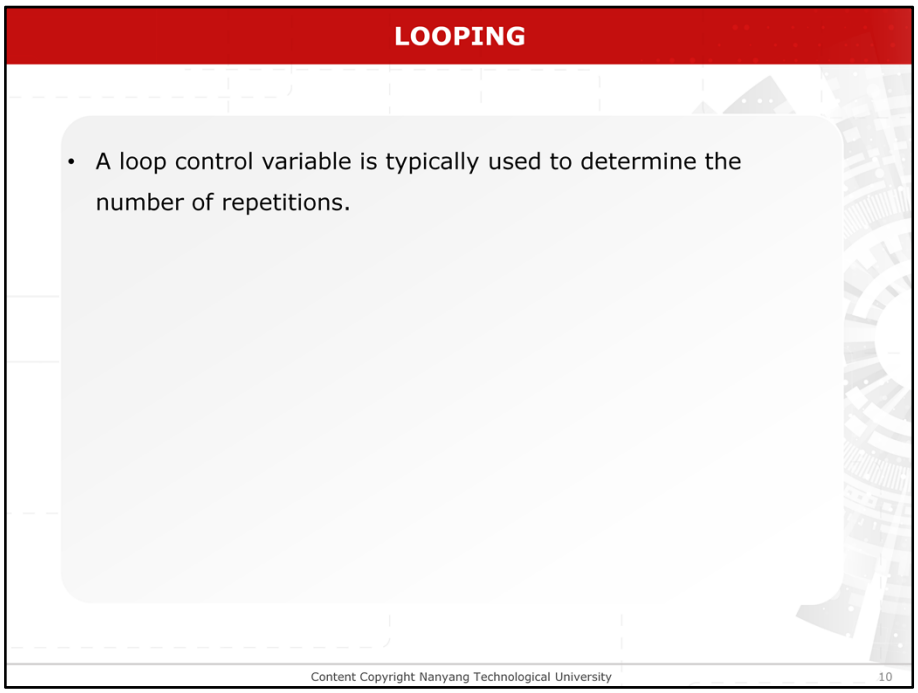
Basically, there are two types of loops: *counter-controlled loops* and *sentinel-controlled loops*.



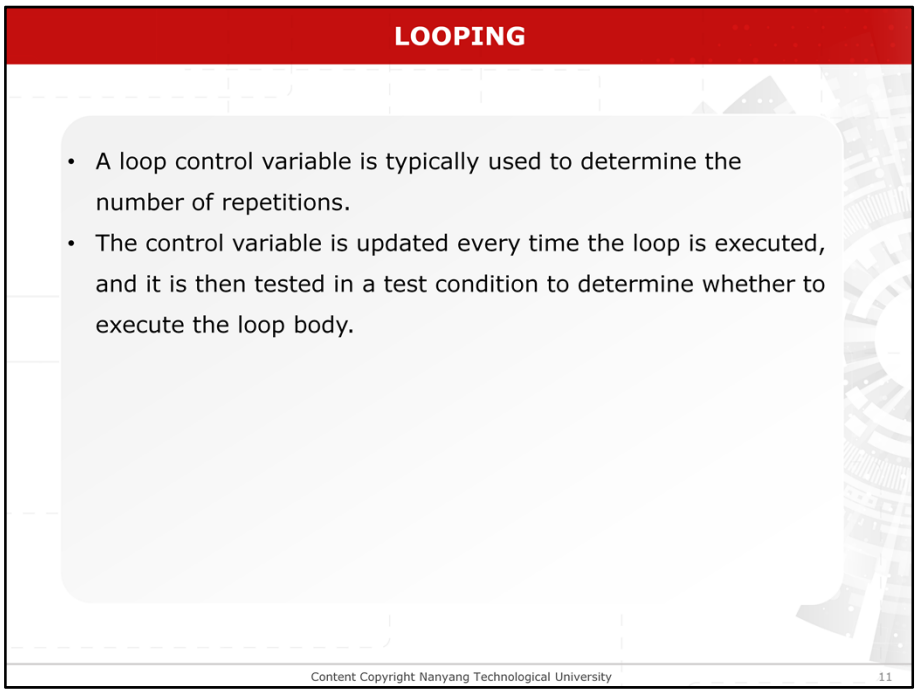
In a counter-controlled loop, the loop body is repeated for a specified number of times, and the number of repetitions is known before the loop begins execution.



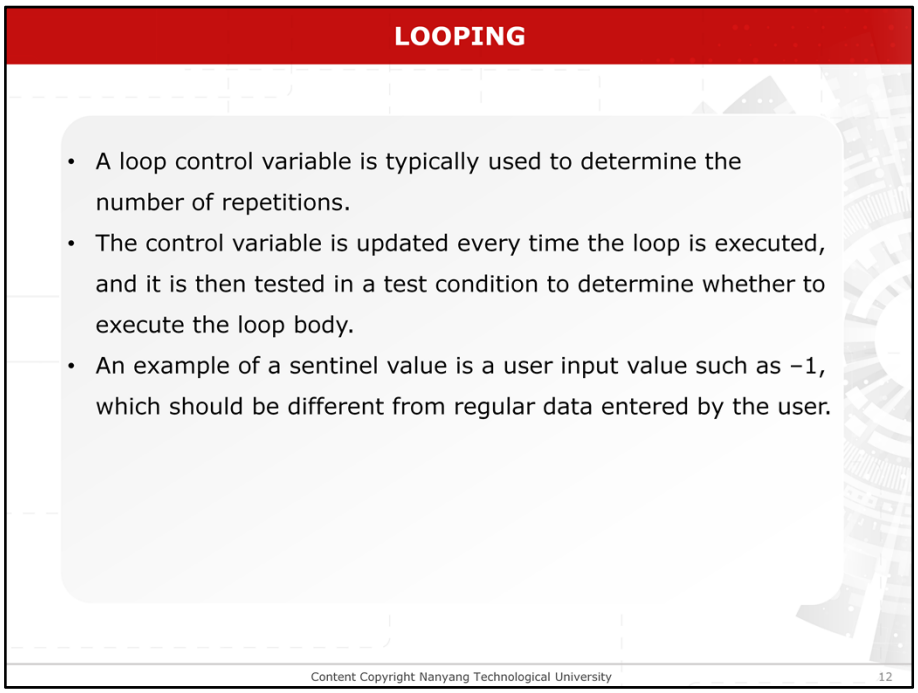
In a sentinel-controlled loop, the number of repetitions is not known before the loop begins execution.



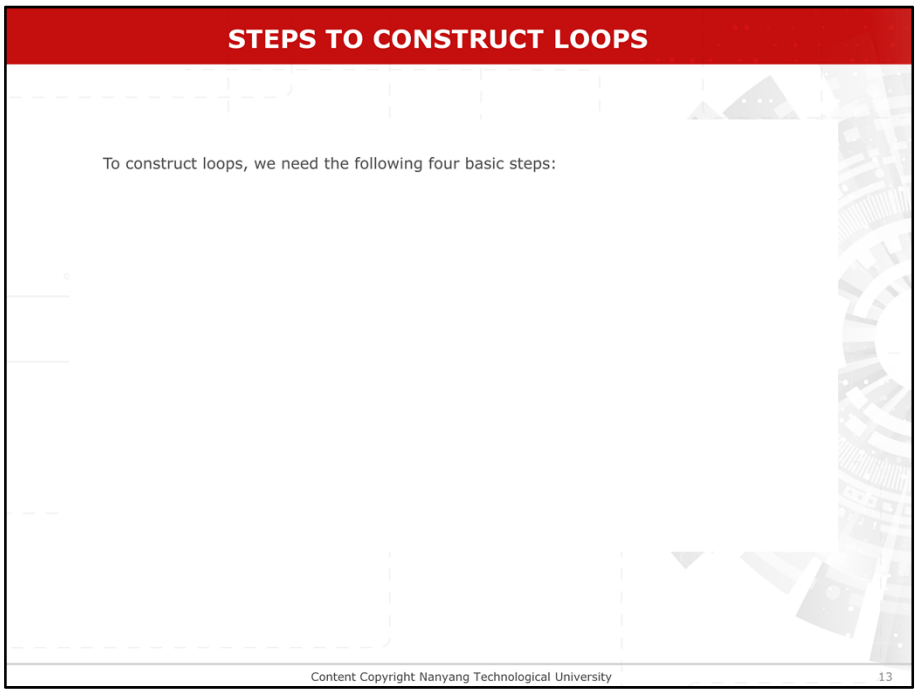
A *loop control variable* is typically used to determine the number of repetitions.



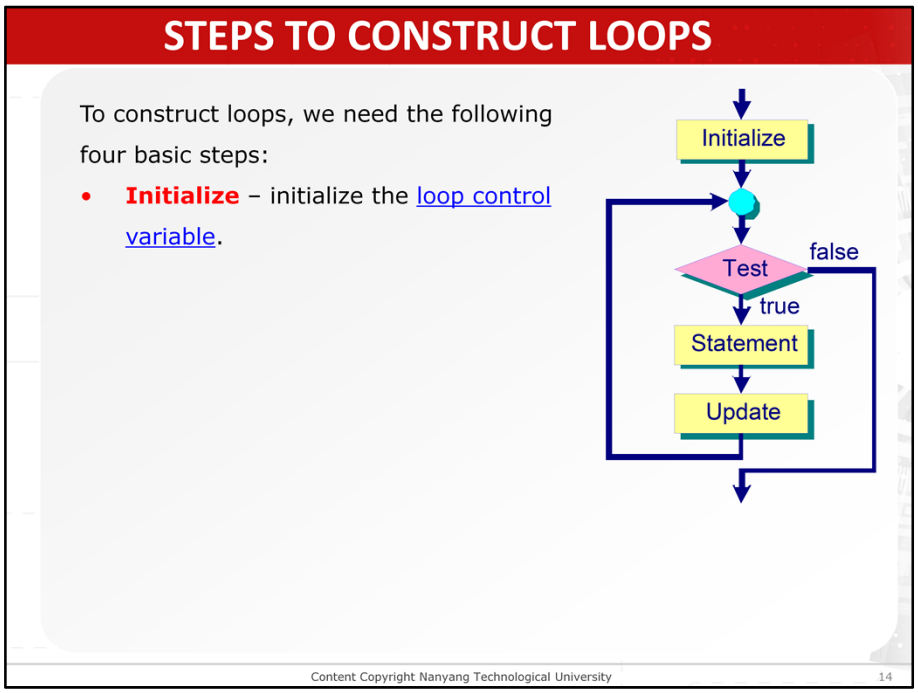
The control variable is updated every time the loop is executed, and it is then tested in a test condition to determine whether to execute the loop body.



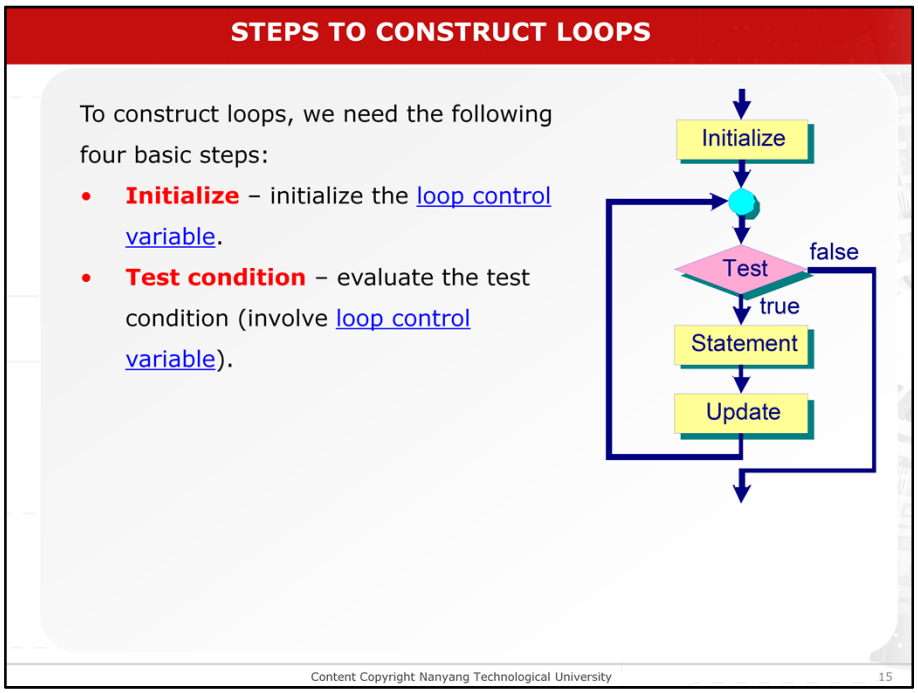
An example of a sentinel value is a user input value such as –1, which should be different from regular data entered by the user.



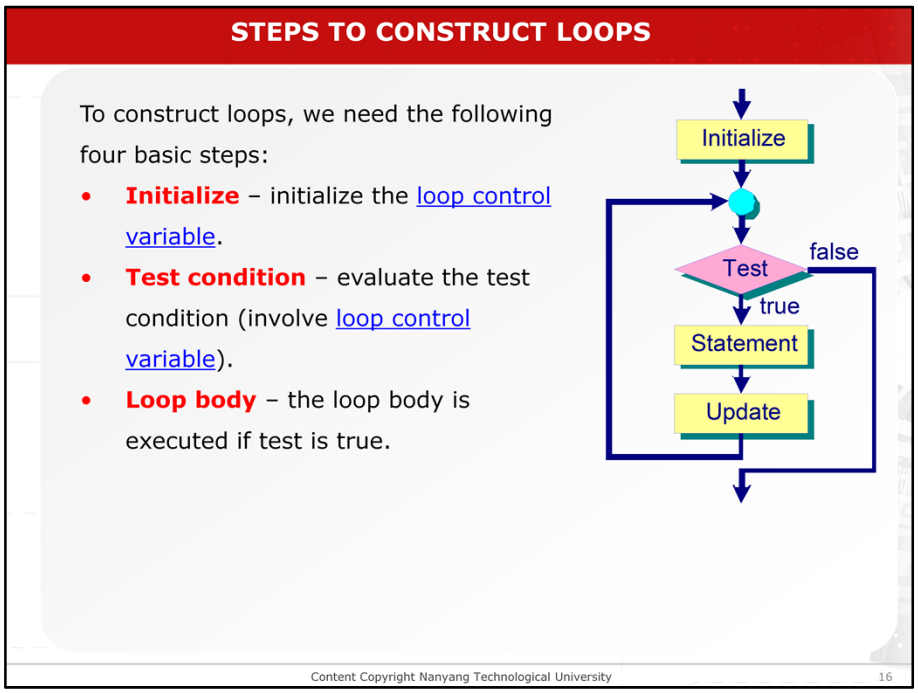
To construct loops, we need the following four basic steps:



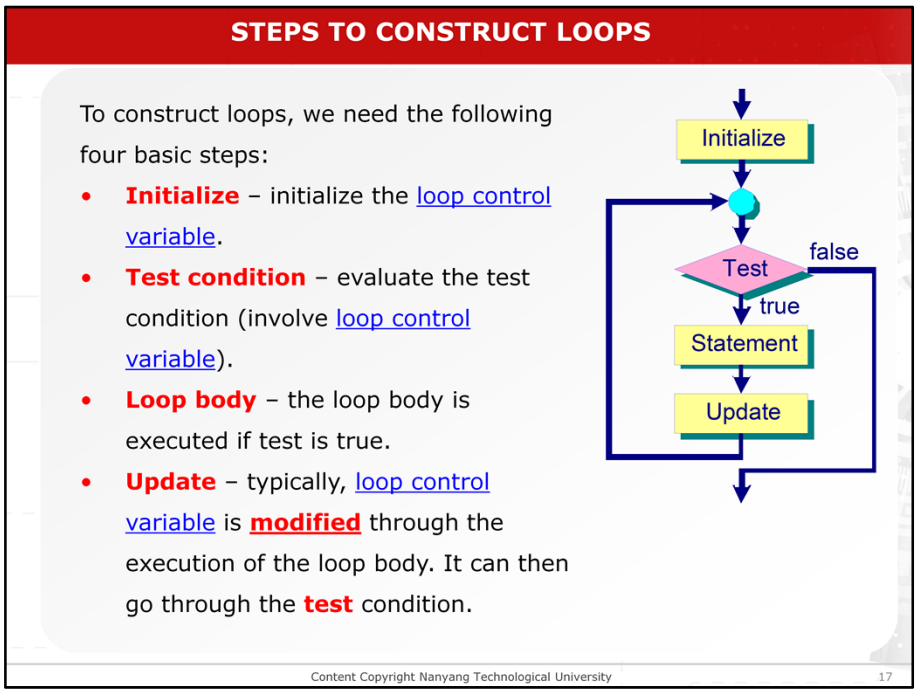
- **Initialize** - This defines and initializes the loop control variable, which is used to control the number of repetitions of the loop.



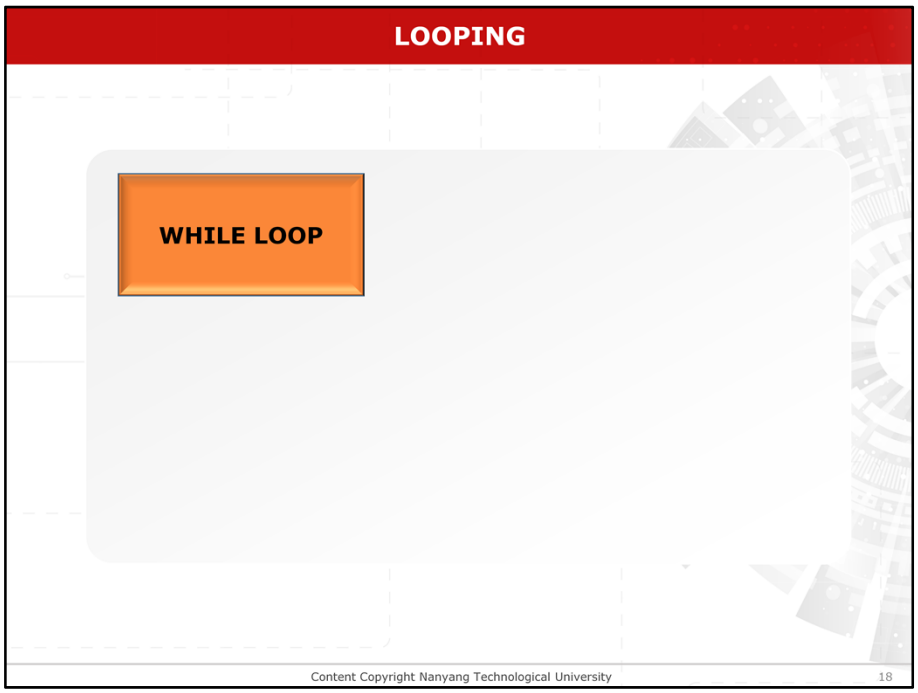
Test - This evaluates the **test** condition. If the **test** condition is true, then the loop body is executed, otherwise the loop is terminated. The **while** loop and **for** loop evaluate the **test** condition at the beginning of the loop, while the **do-while** loop evaluates the **test** condition at the end of the loop.



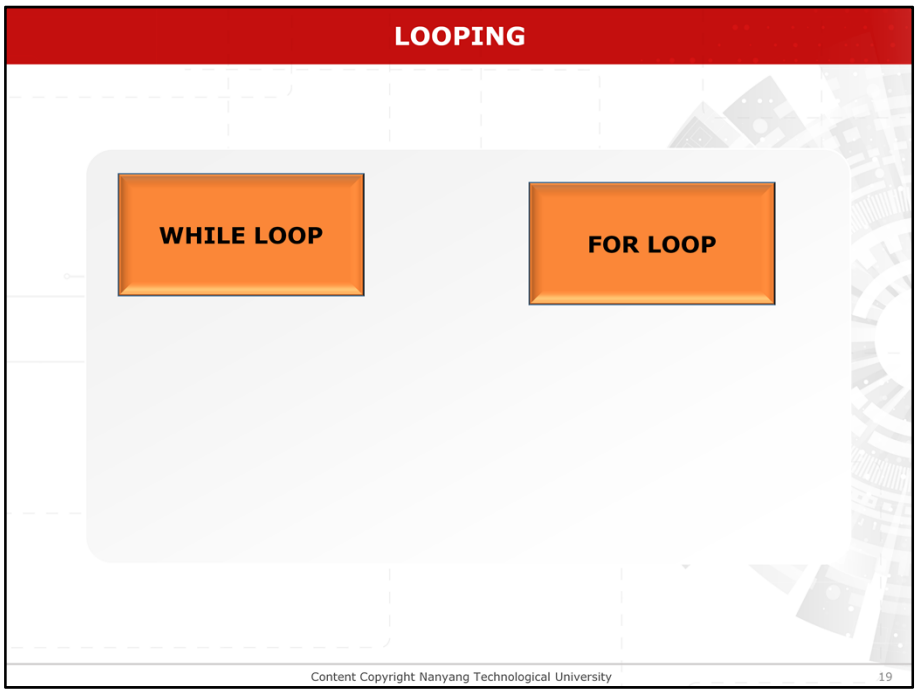
Loop body - The **loop body** is executed if the **test** condition is evaluated to be true. It contains the actual actions to be carried out.



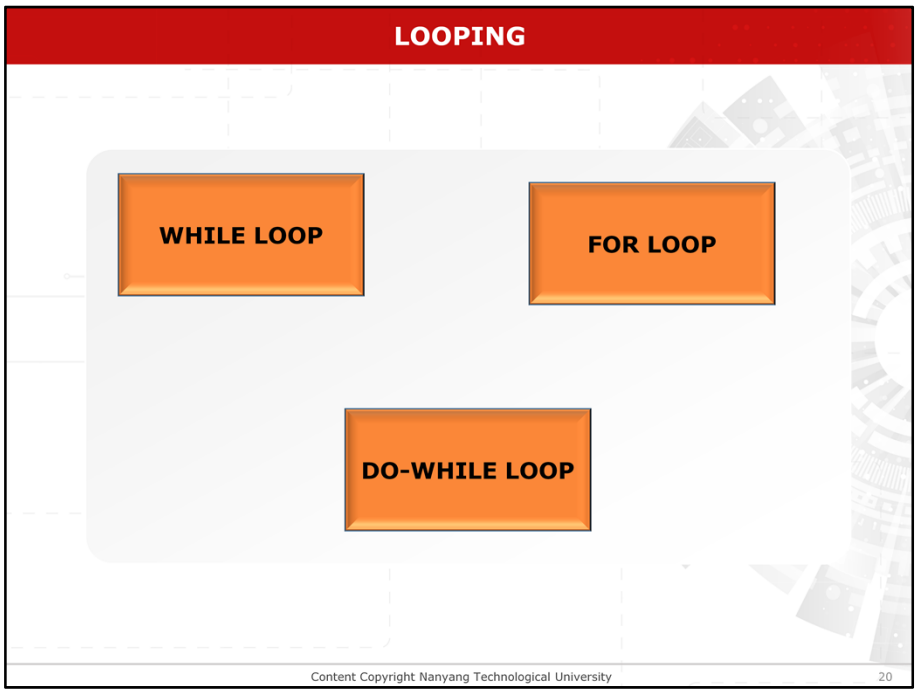
Update - The **test** condition typically involves the loop control variable that should be modified each time through the execution of the loop body. The loop control variable will go through the **test** condition again to determine whether to repeat the loop body again.



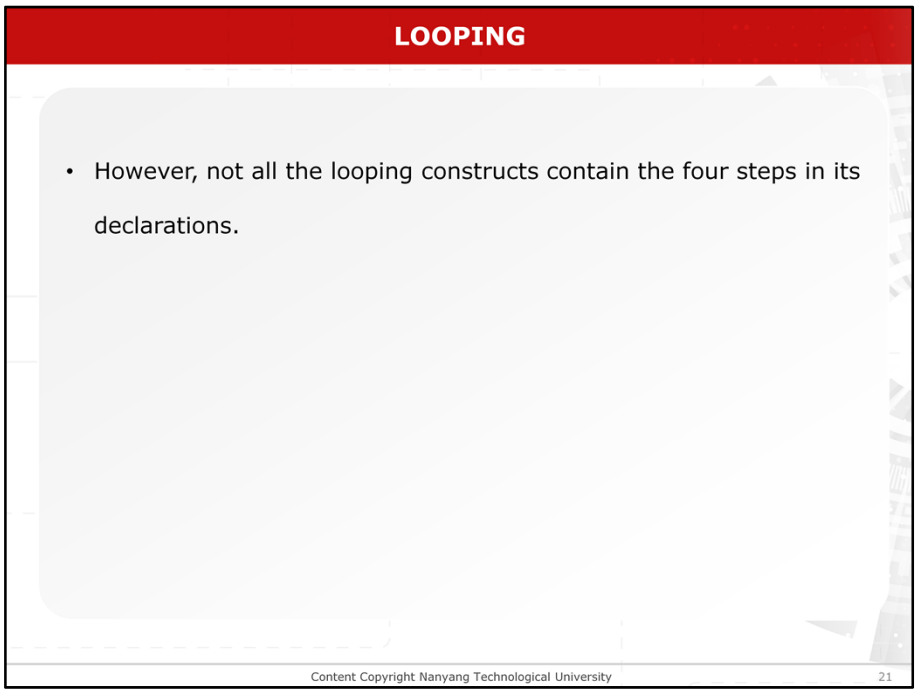
We can use one of the three looping constructs, namely the **while** loop,


LOOPING
WHILE LOOP
FOR LOOP
Content Copyright Nanyang Technological University
19

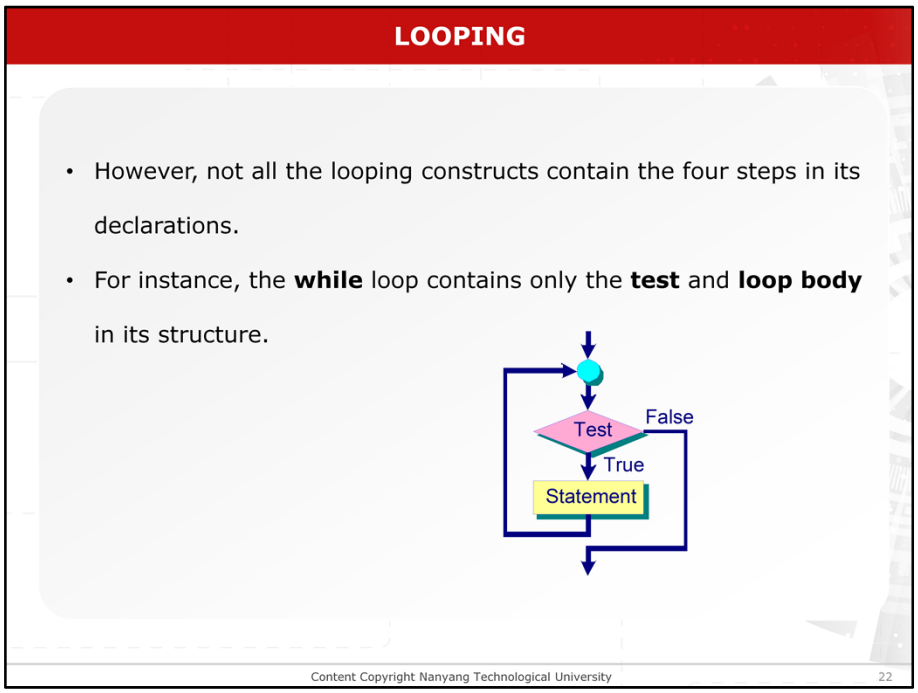
the **for** loop



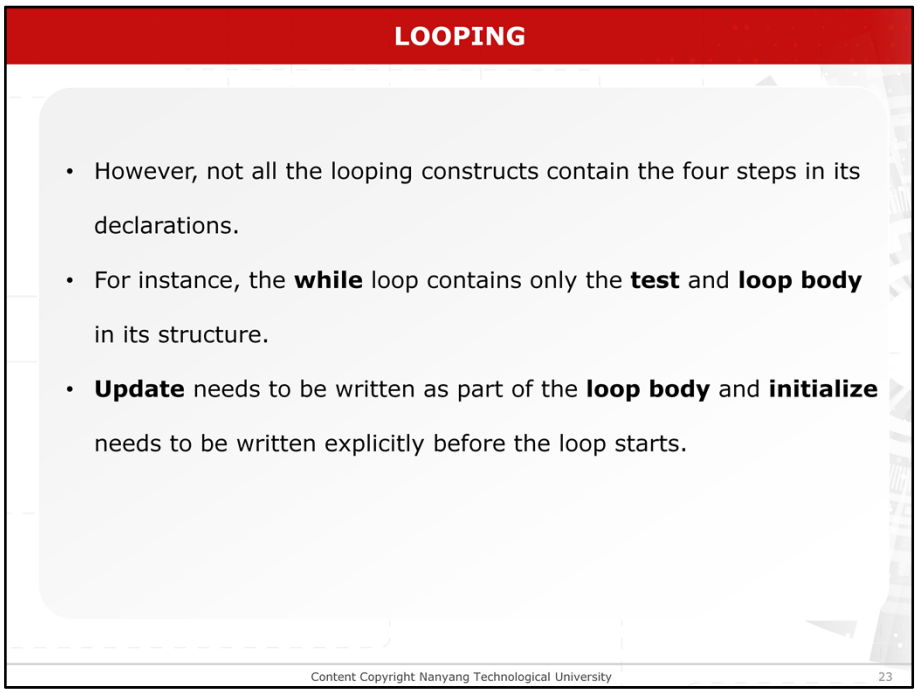
and the **do-while** loop, for constructing loops



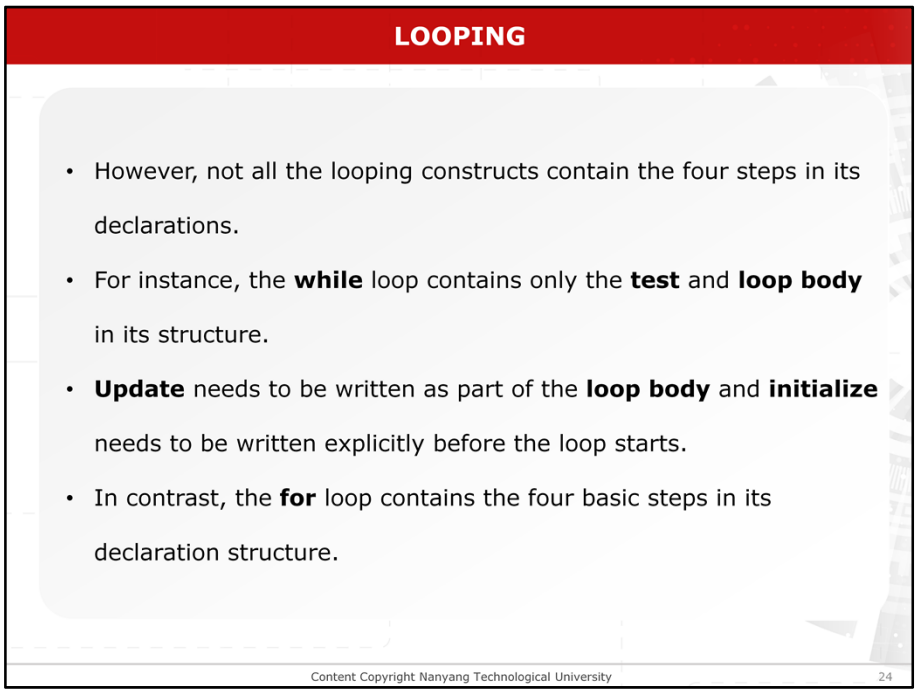
However, not all the looping constructs contain the four steps in its declarations.



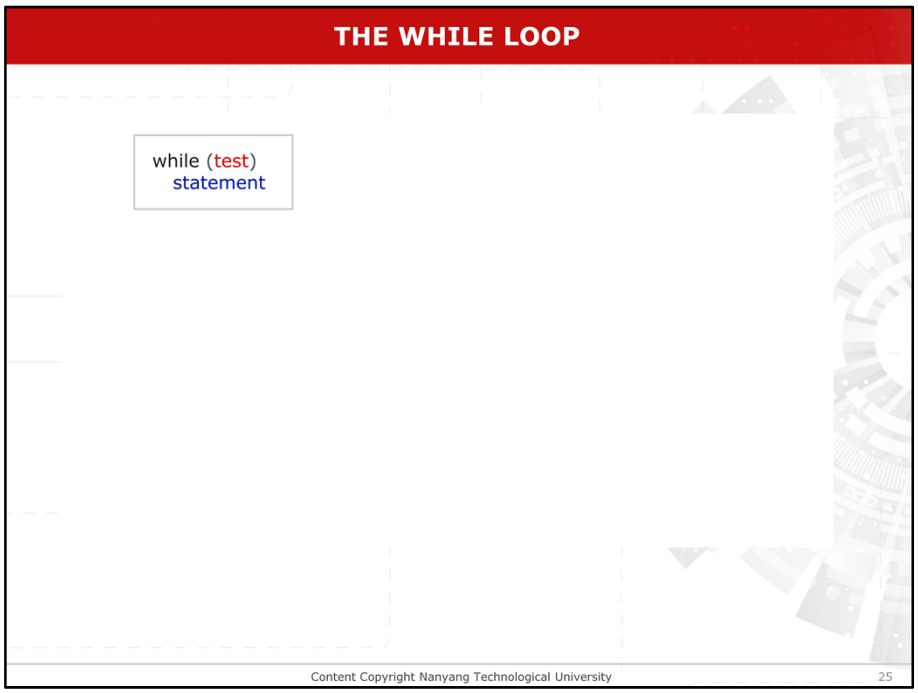
For instance, the **while** loop contains only the **test** and **loop body** in its structure.



Update needs to be written as part of the **loop body**, and **initialize** needs to be written explicitly before the loop starts.

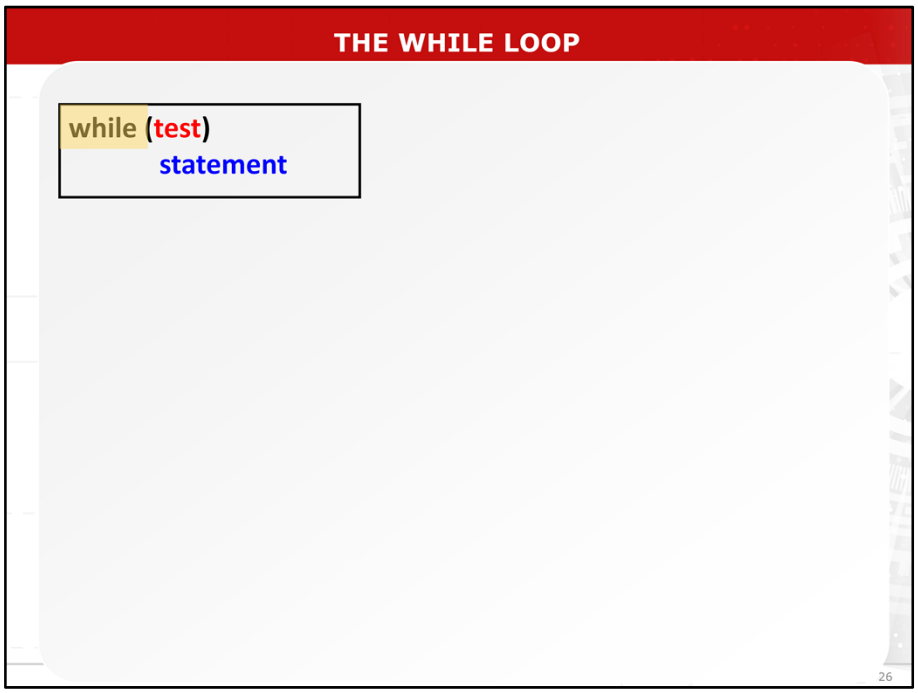


In contrast, the **for** loop contains the four basic steps in its declaration structure.

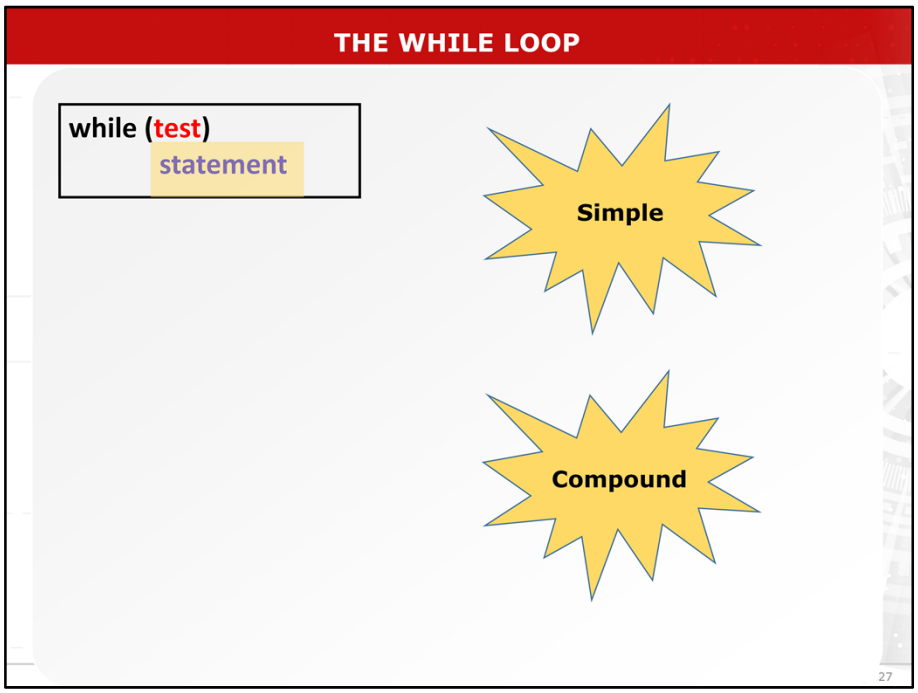


The while Loop

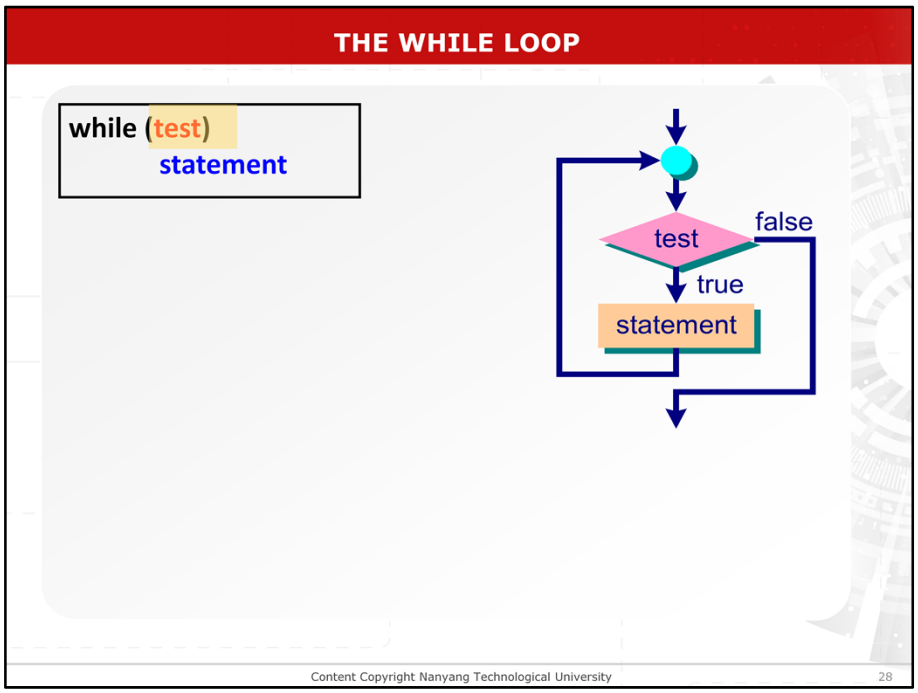
The format of the **while** statement is shown here.



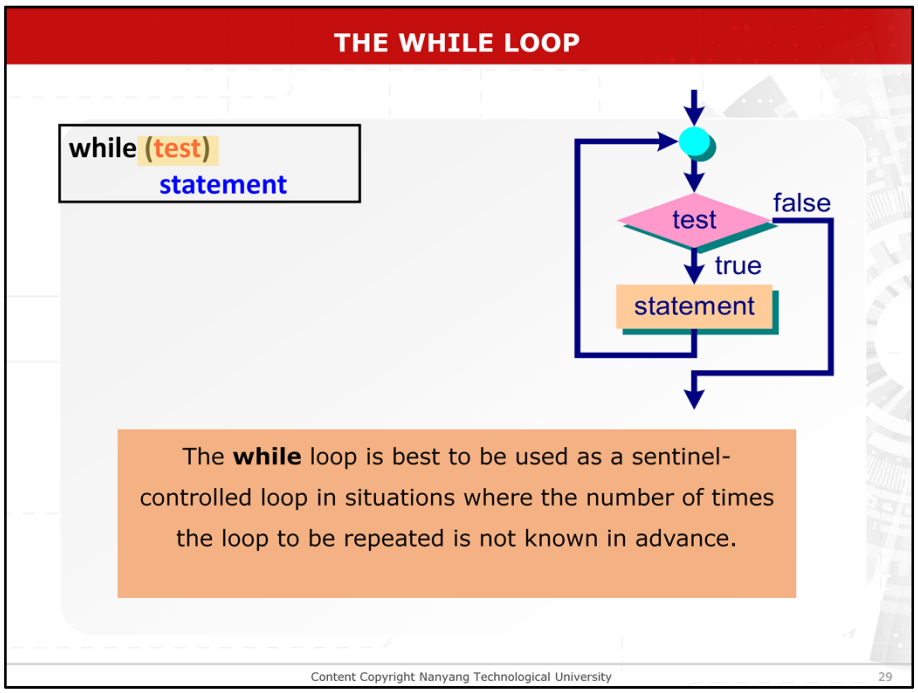
where **while** is a reserved keyword.



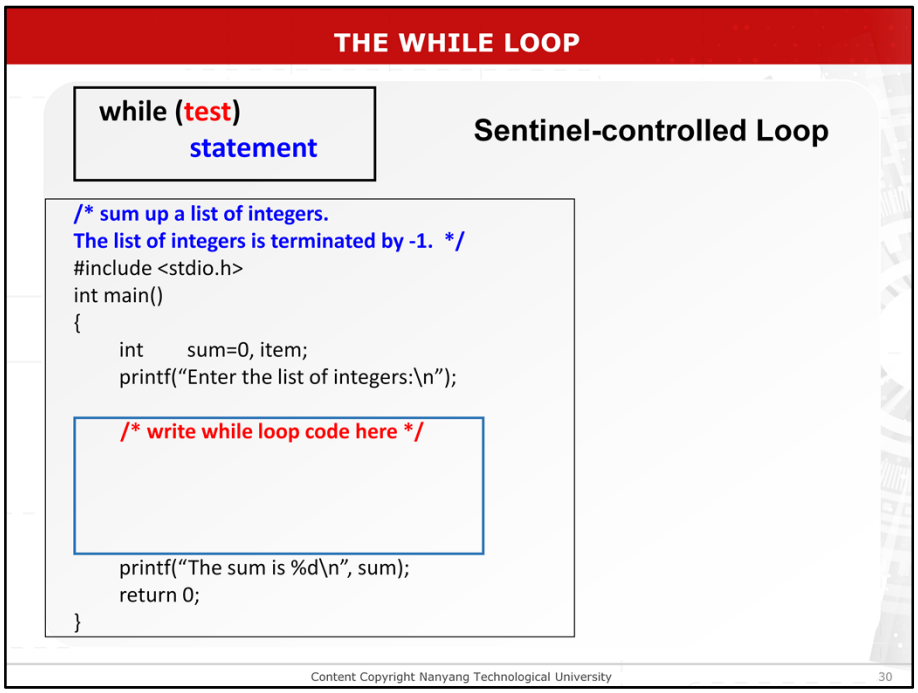
statement can be a simple statement terminated by a semicolon or a compound statement enclosed by **braces.**



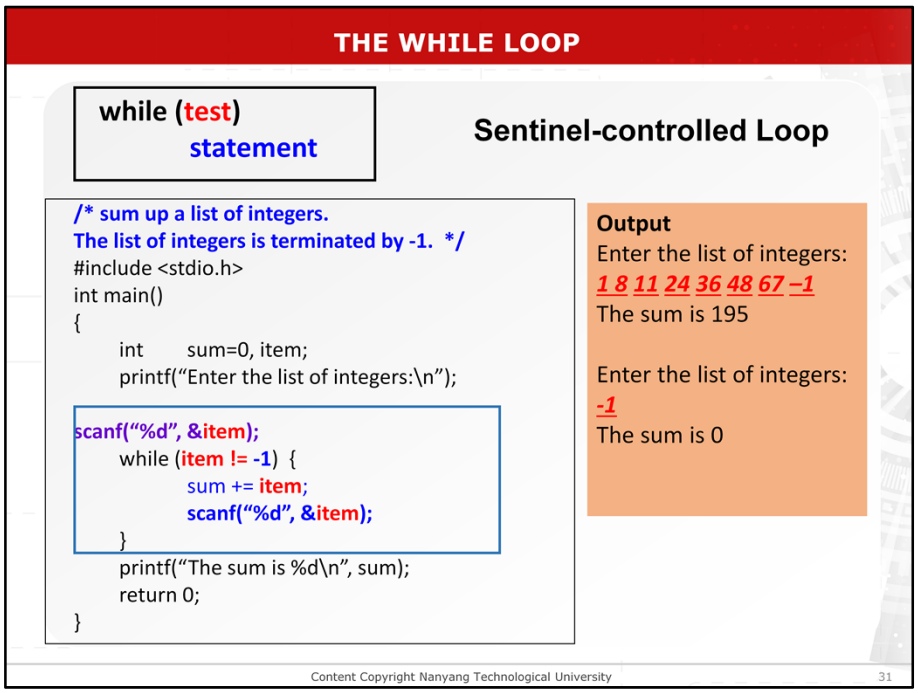
The **while** statement is executed by evaluating the **test** condition.
If the result is true, then **statement** is executed. Control is then transferred back to the beginning of the **while** statement, and the process repeats again. This looping continues until the **test** condition finally becomes false. When the **test** condition is false, the loop terminates and the program continues execute the next sequential statement.



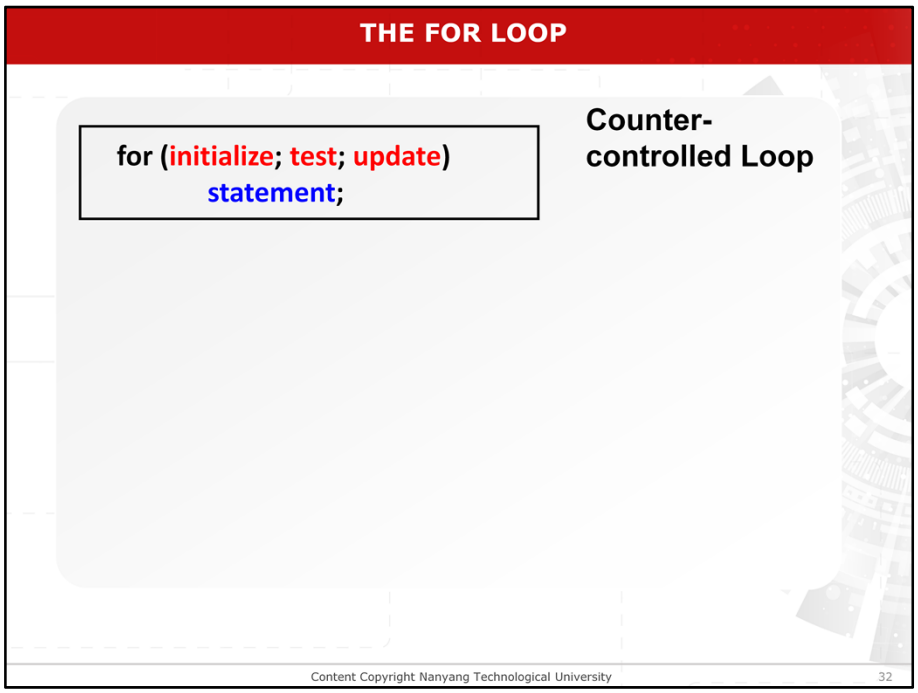
The **while** loop is best to be used as a sentinel-controlled loop in situations where the number of times the loop to be repeated is not known in advance.



In the example program, it aims to sum up a list of integer numbers, and the list of numbers is terminated by -1.

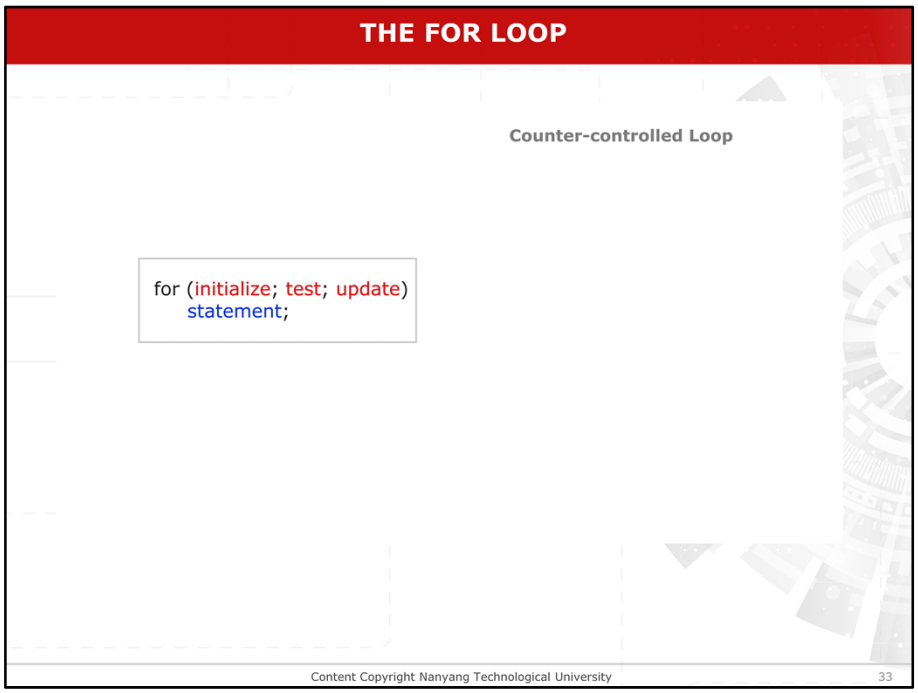


In the example program, it aims to sum up a list of integer numbers, and the list of numbers is terminated by -1. The program does not know how many data items are to be read at the beginning of the program. However, it will keep on reading the data until the user input is –1 which is the sentinel value. The **scanf()** statement reads in the first number and stores it in the variable **item**. Then, the execution of the **while** loop begins. If the initial **item** value is not –1, then the statements in the braces are executed. The **item** value is first added to another variable **sum**. Another **item** value is then read in from the user, and the control is transferred back to the **test** expression **(item != -1)** for evaluation. This process repeats until the **item** value becomes –1.

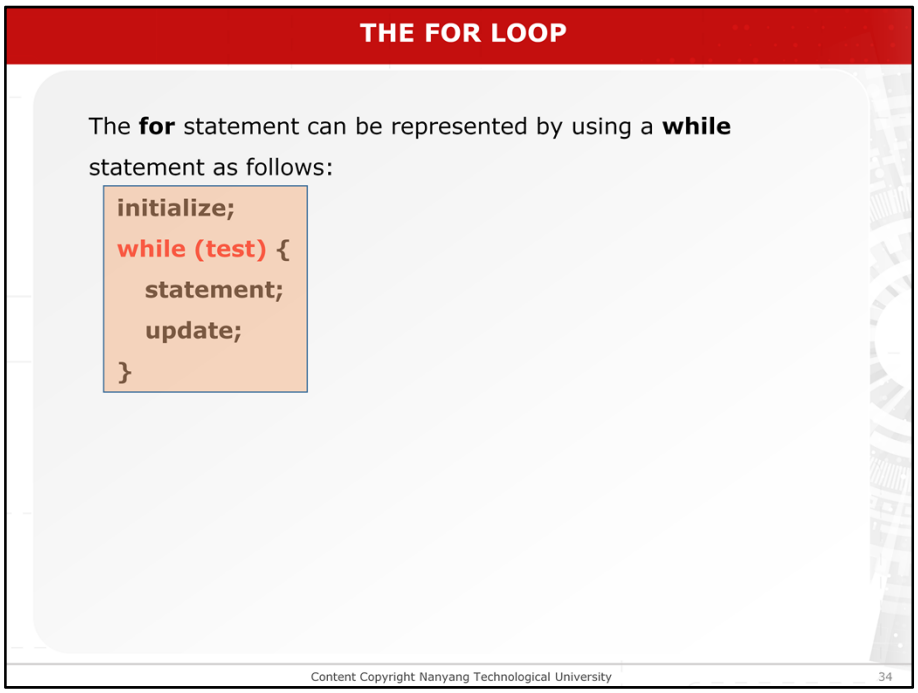


The for Loop

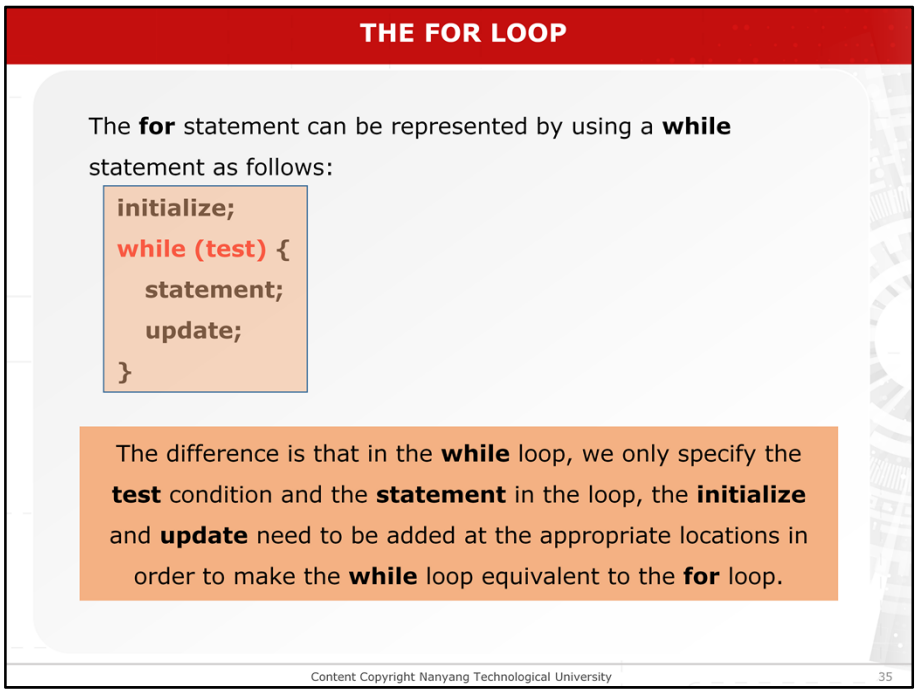
The **for** statement allows us to repeat a sequence of statements for a specified number of times which is known in advance. The **for** loop is mainly used as a *counter-controlled loop*. The format of the **for** statement is shown here, where **for** is a reserved keyword. **statement** can be a simple statement terminated by a semicolon or a compound statement enclosed by **{}**. **initialize** is usually used to set the initial value of one or more loop control variables. Generally, **test** is a relational expression to control iterations. **update** is used to update some loop control variables before repeating the loop.



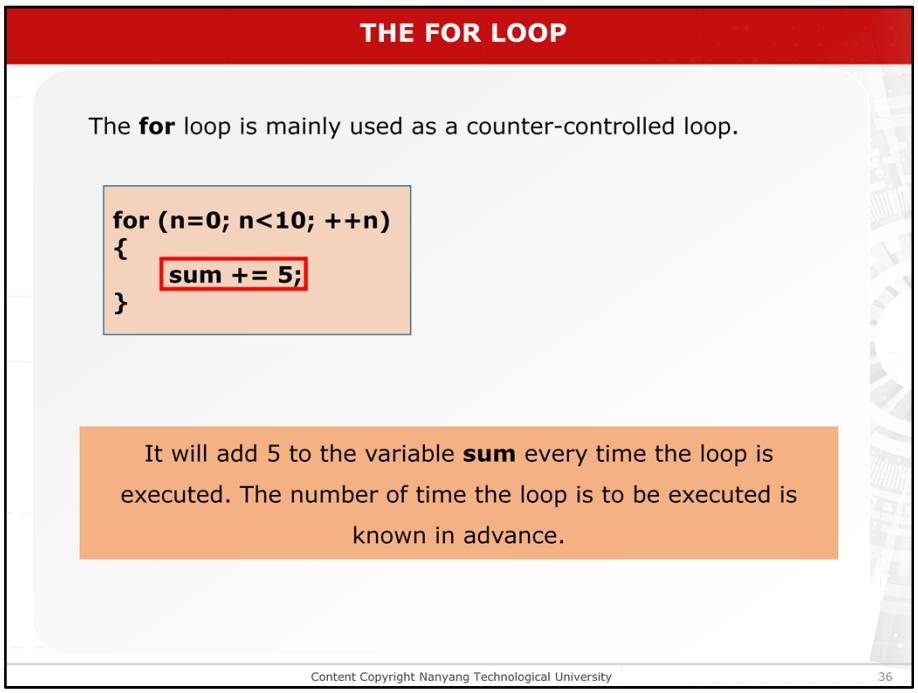
In the **for** loop, **initialize** is first evaluated. The **test** condition is then evaluated. If the **test** condition is true, then the **statement** and **update** expression are executed. Control is then transferred to the **test** condition, and the loop is repeated again if the **test** condition is true. If the **test** condition is false, then the loop is terminated. The control is then transferred out of the **for** loop to the next sequential statement.



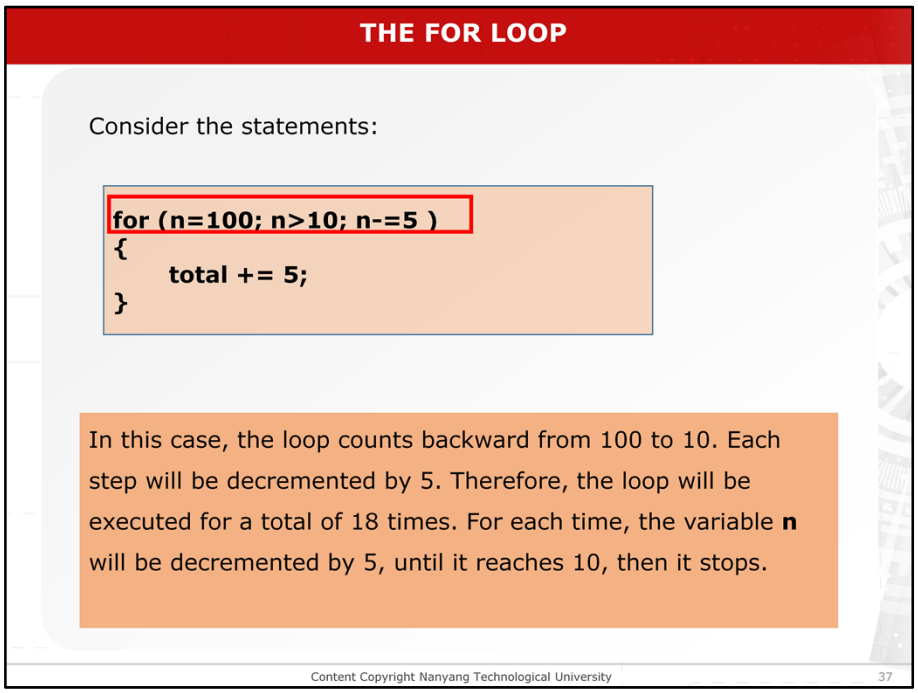
The **for** statement can be represented by using a **while** statement as shown here.



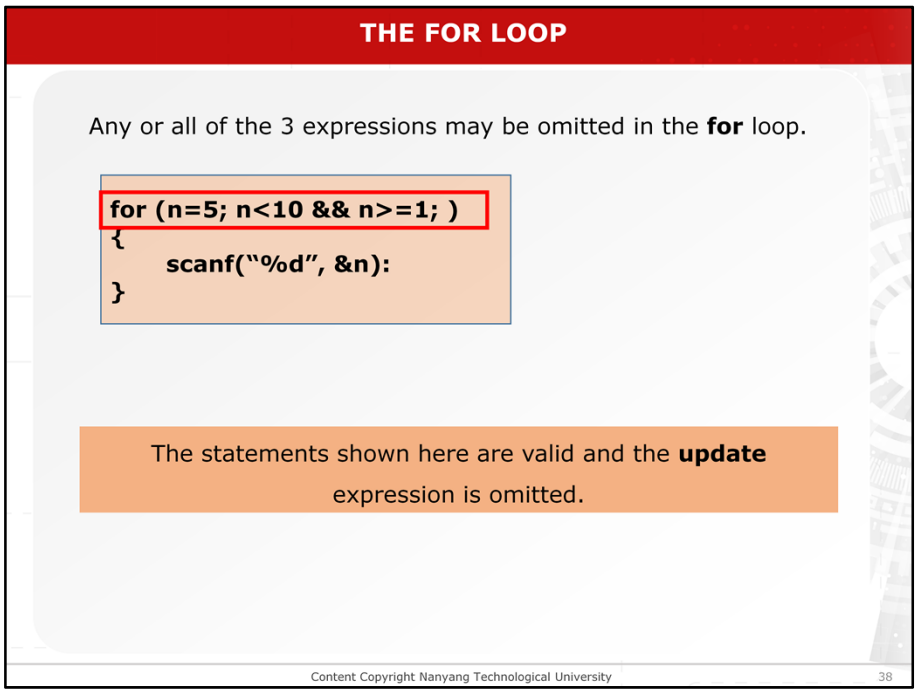
The difference is that in the **while** loop, we only specify the **test** condition and the **statement** in the loop, the **initialize** and **update** need to be added at the appropriate locations in order to make the **while** loop equivalent to the **for** loop.



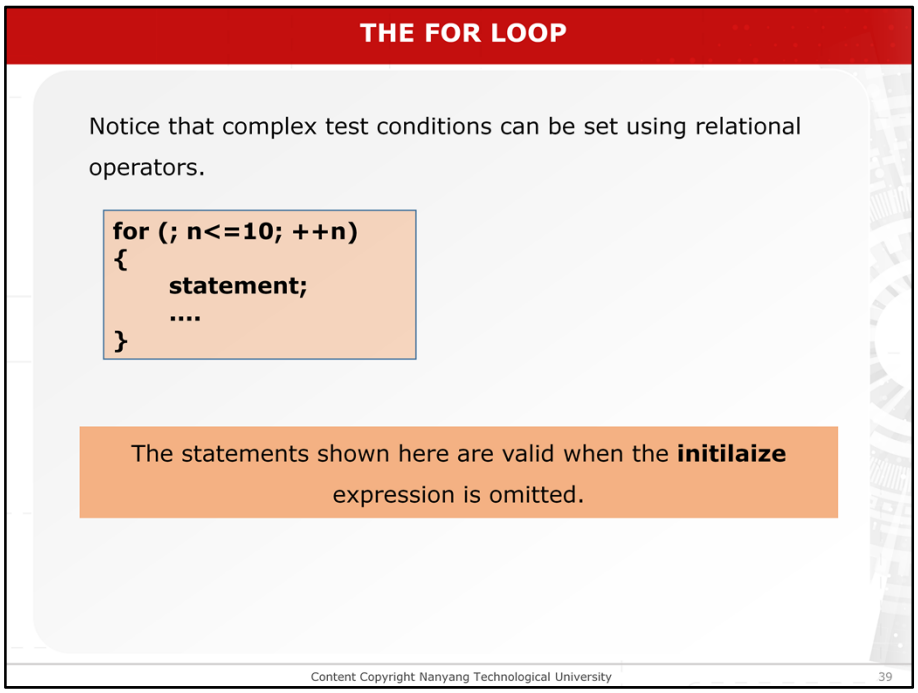
The **for** loop is mainly used as a counter-controlled loop. For example, the code shown here willl add 5 to the variable **sum** every time the loop is executed. The number of time the loop is to be executed is known in advance.



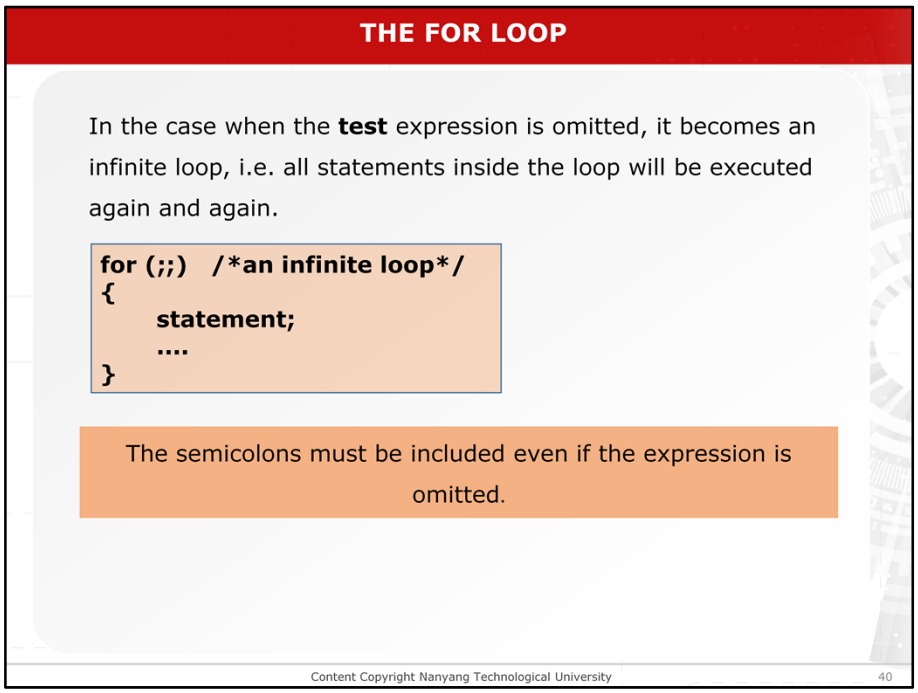
Consider the statements shown here. In this case, the loop counts backward from 100 to 10. Each step will be decremented by 5. Therefore, the loop will be executed for a total of 18 times. For each time, the variable **n** will be decremented by 5, until it reaches 10, then it stops.



Any or all of the 3 expressions may be omitted in the **for** loop. For example, the statements shown here are valid and the **update** expression is omitted.

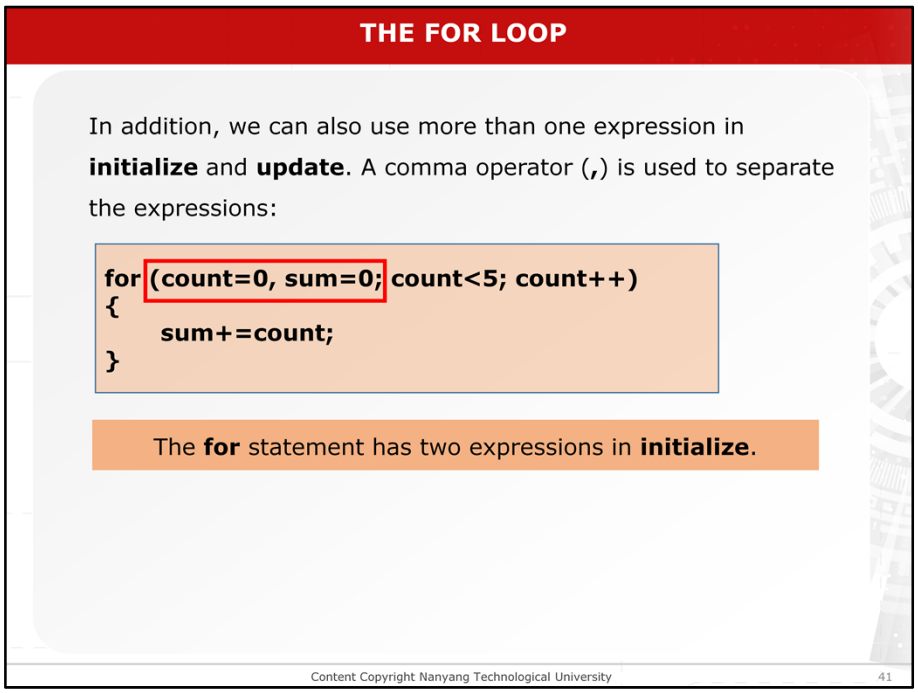


Notice that complex **test** conditions can be set using relational operators. The statements shown here are also valid when the **initialize** expression is omitted.



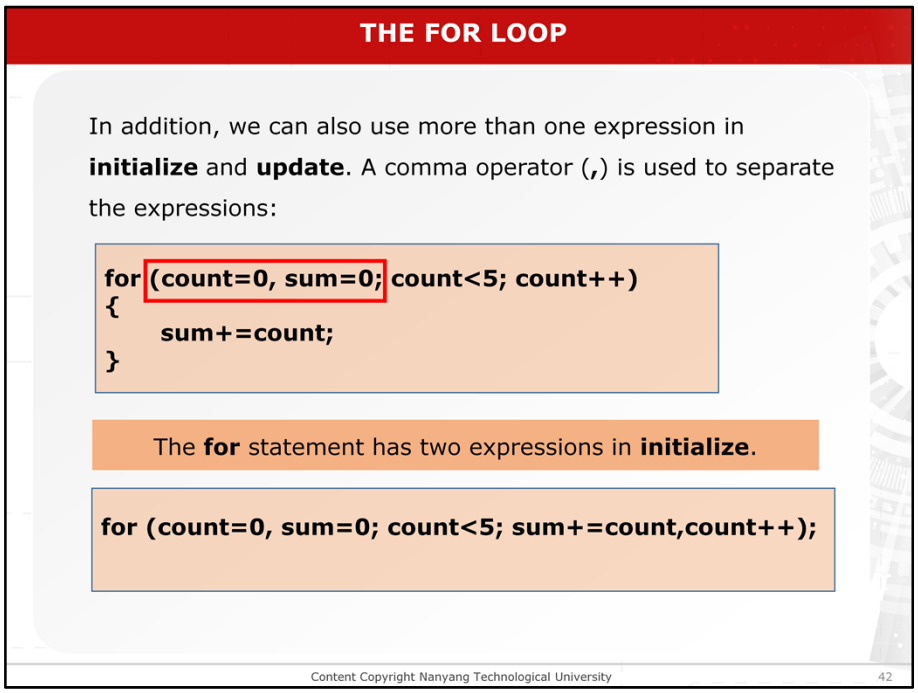
In the case when the **test** expression is omitted, it becomes an infinite loop, i.e. all statements inside the loop will be executed again and again.

Notice that the semicolons must be included even if the expression is omitted.

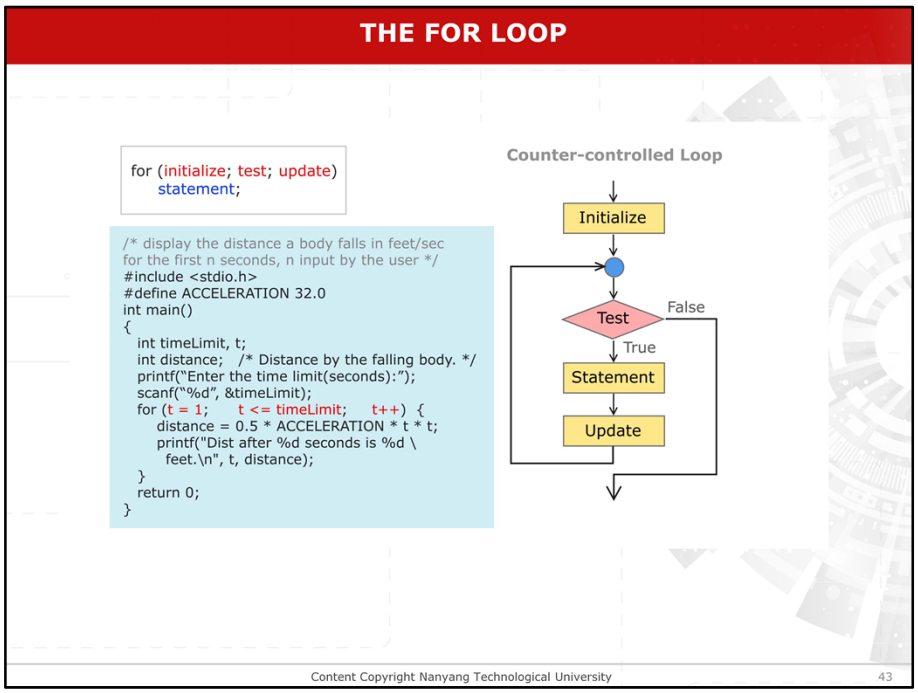


In addition, we can also use more than one expression in **initialize** and **update**. A comma operator (**,**) is used to separate the expressions:

The **for** statement has two expressions in **initialize**.

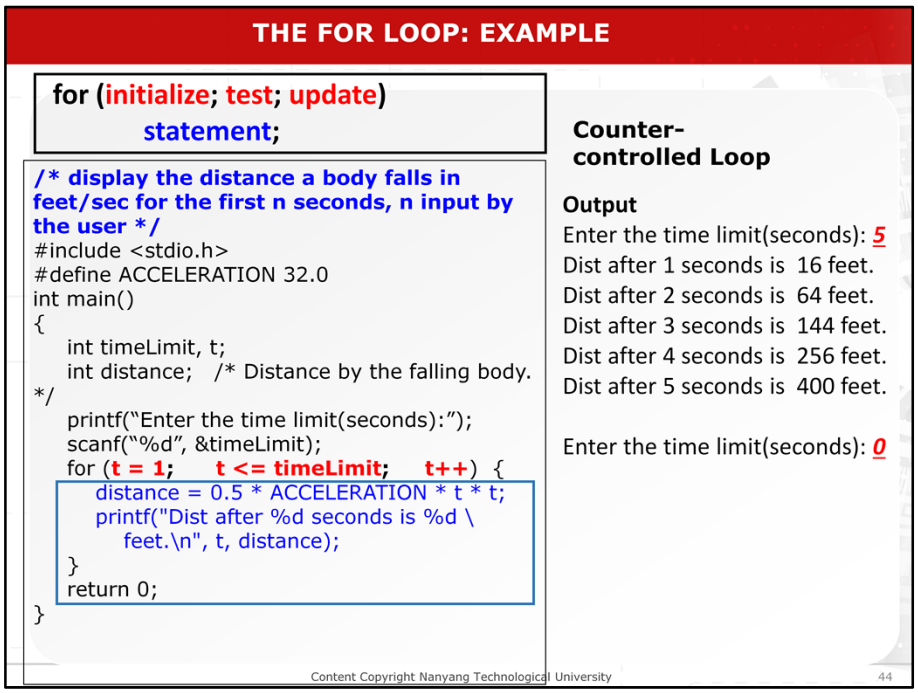


We can also have the **for** statement to perform the above task as shown here. For the **for** loop, the loop body is null (**;**) which does nothing.



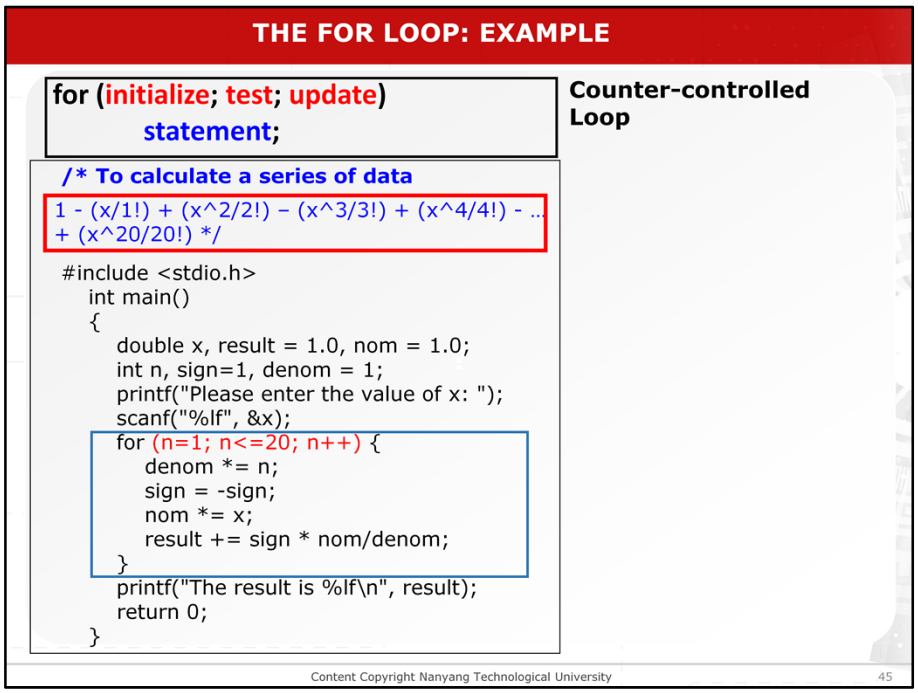
The for Loop

In the example program, it aims to display the distance a body falls in feet/sec for the first n seconds, where n is the user input.

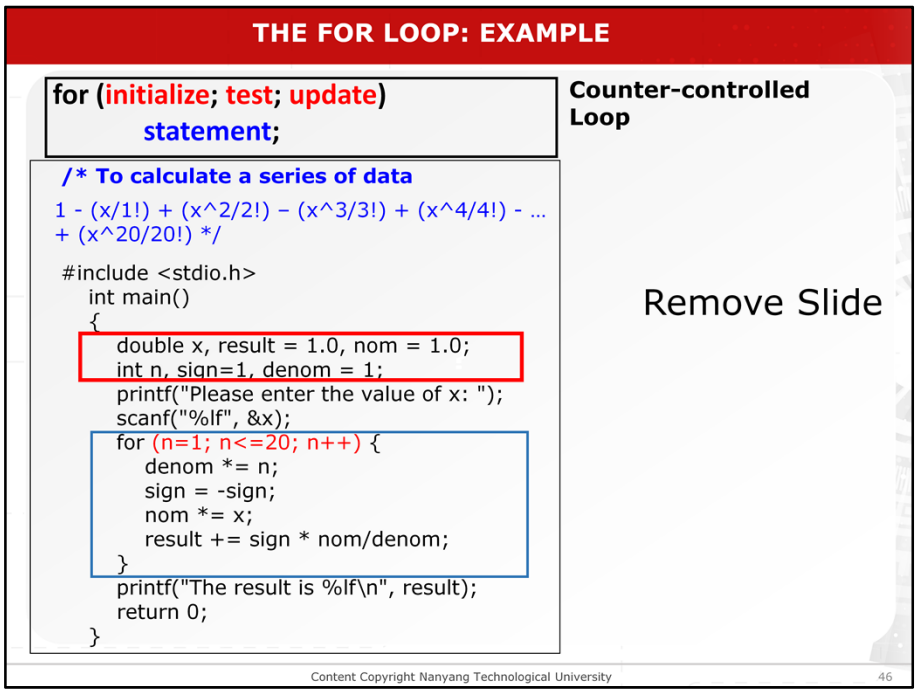


The for Loop: Example

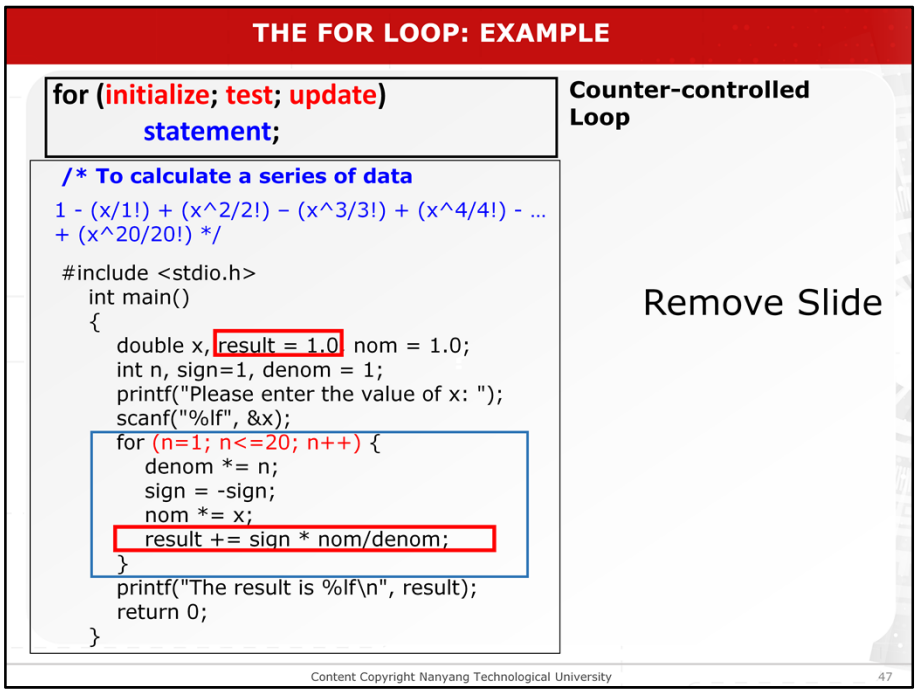
The program reads in the time limit and stores it in the variable **timeLimit**. It then uses the **for** loop as a counter-controlled loop. The loop control variable **t** is used to control the number of repetitions in the loop. The variable **t** is initialized to 1. In the loop body, the distance calculation formula is used to compute the distance. The loop control variable **t** is also incremented by 1 every time the loop body finishes executing. The loop will stop when **t** equals to **timeLimit**.



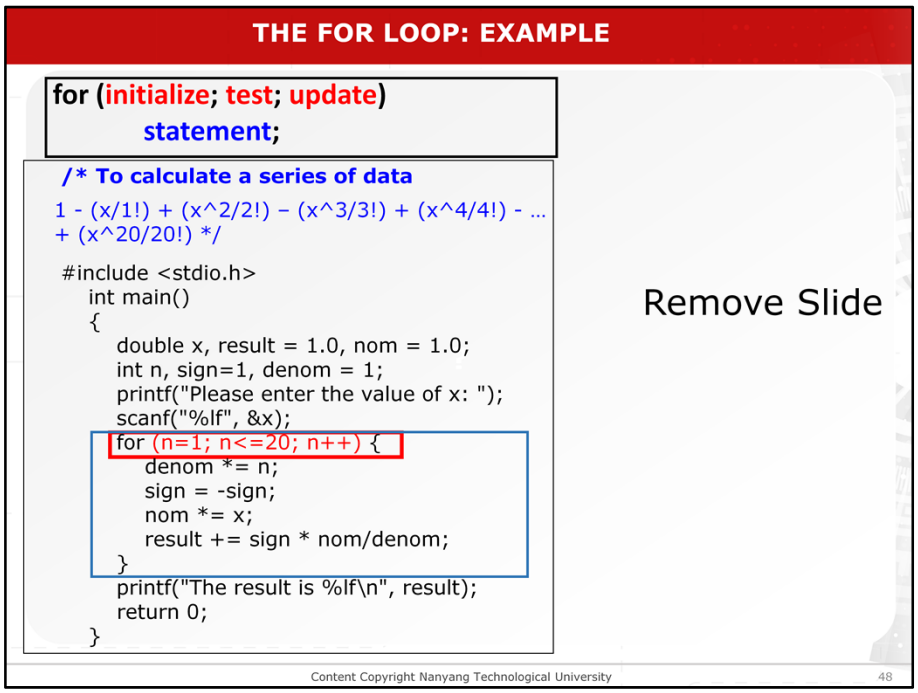
Another example of using the **for** loop is to calculate a series of data as shown here. Before we write the program, we observe that each component of the series consists of three parts: the part involving **x**, the part on the factorial, and the sign.



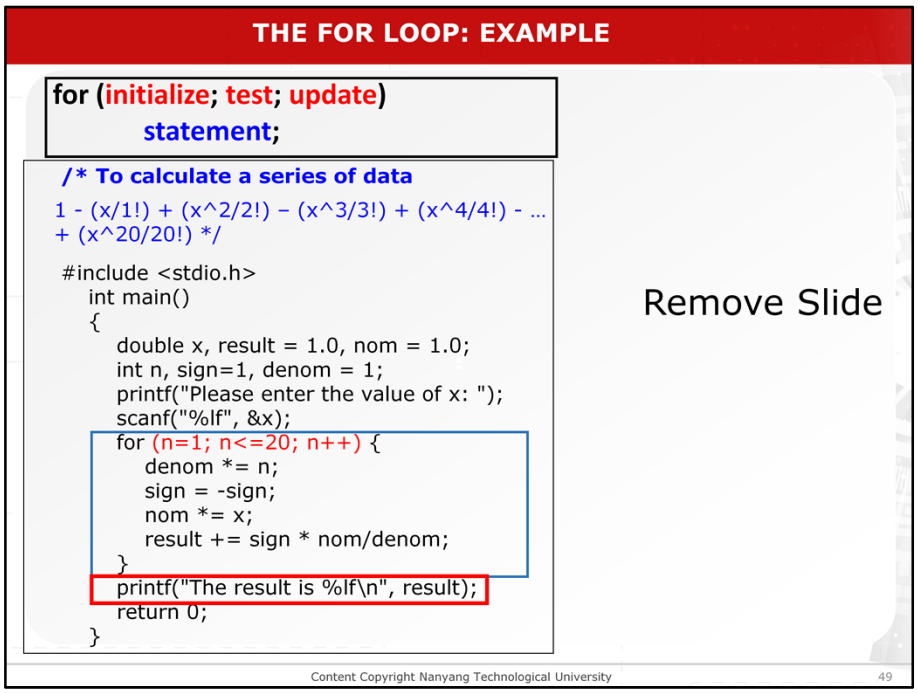
Therefore, we create three variables, namely **nom**, **denom** and **sign** to store the data of each part of the component. In addition, the variable **result** is used to calculate the value of the series for each component of the **for** loop. The variable **result** is initialized to 1.0. The loop control variable **n** is used to control the loop execution. It is initialized to 1. It stops execution after 20 iterations when **n** equals to 21.



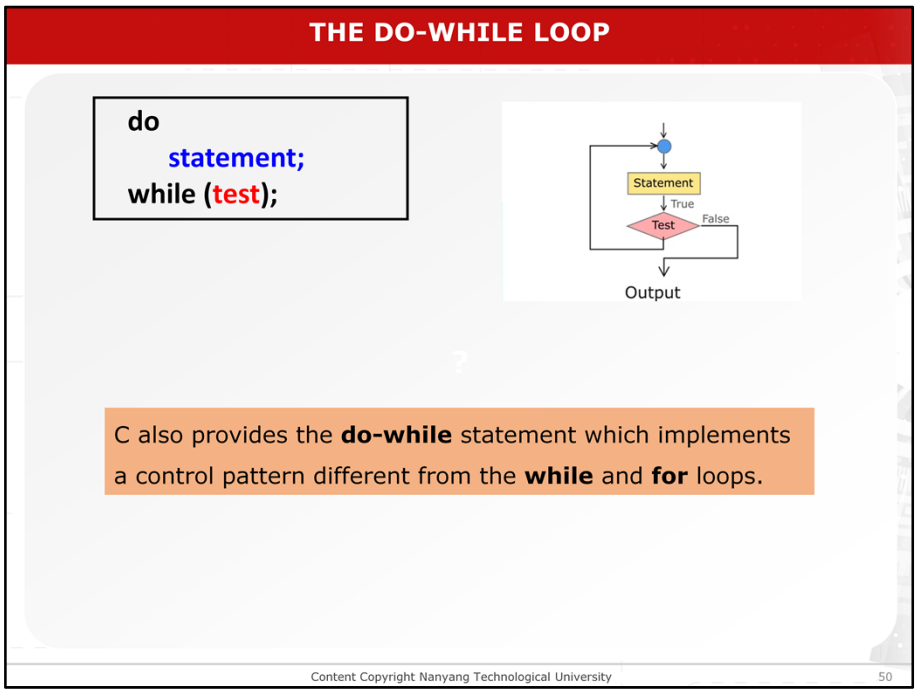
. In addition, the variable **result** is used to calculate the value of the series for each component of the **for** loop. The variable **result** is initialized to 1.0



The loop control variable **n** is used to control the loop execution. It is initialized to 1. It stops execution after 20 iterations when **n** equals to 21

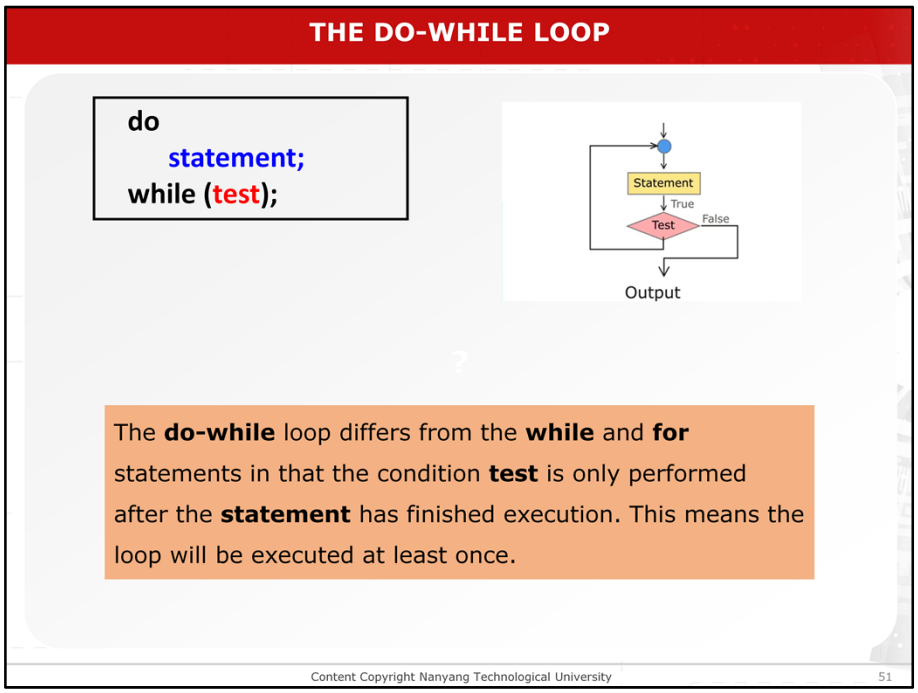


and prints the result.

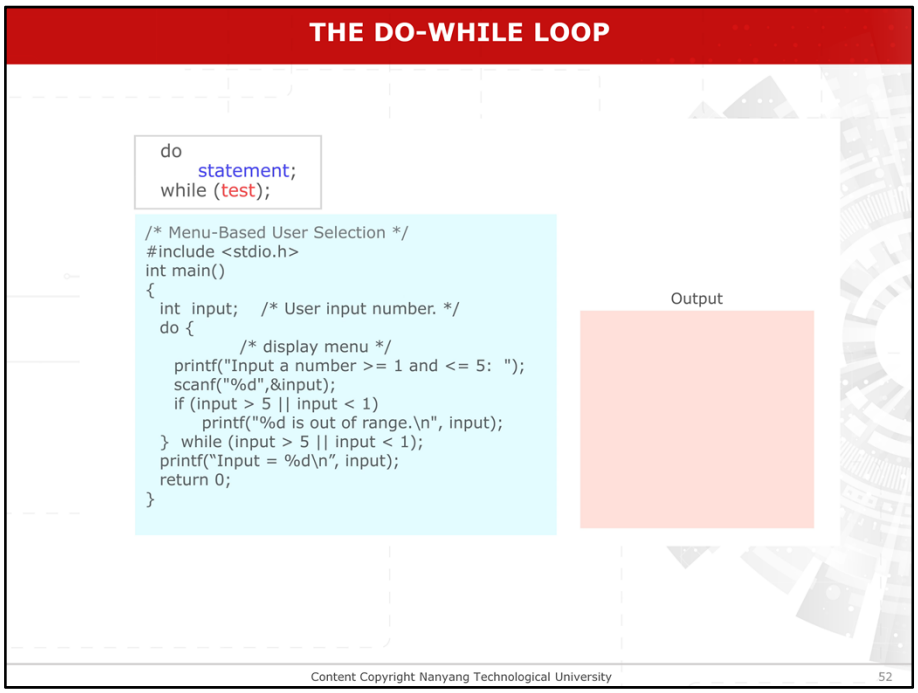


The do-while Loop

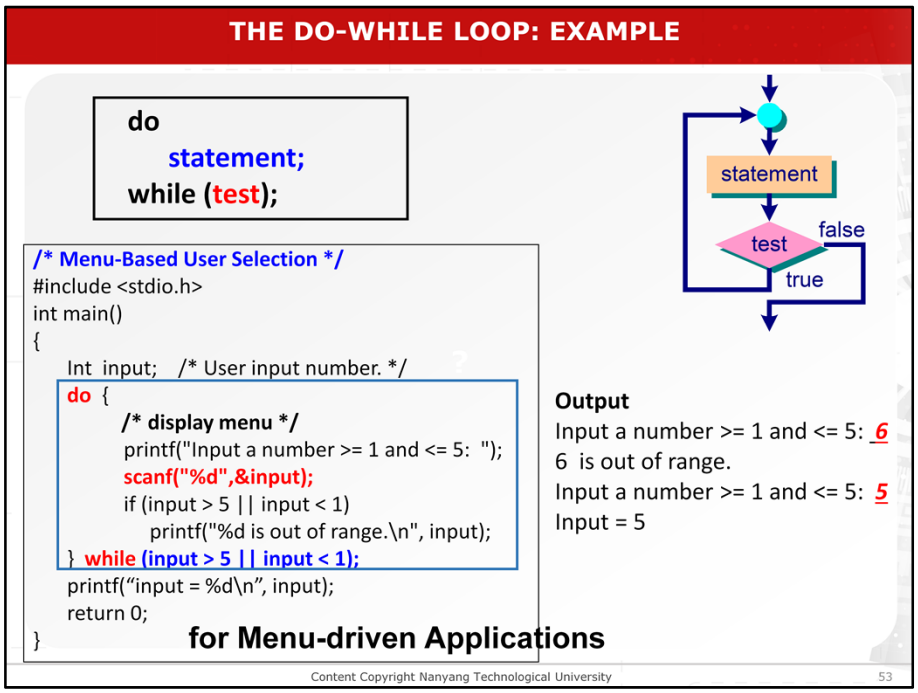
The body of a **do-while** loop is executed at least once. Its general format is SHOWN HERE. Both the **while** and the **for** loops test the condition prior to executing the loop body. C also provides the **do-while** statement which implements a control pattern different from the **while** and **for** loops.



The **do-while** loop differs from the **while** and **for** statements in that the condition **test** is only performed after the **statement** has finished execution. This means the loop will be executed at least once.

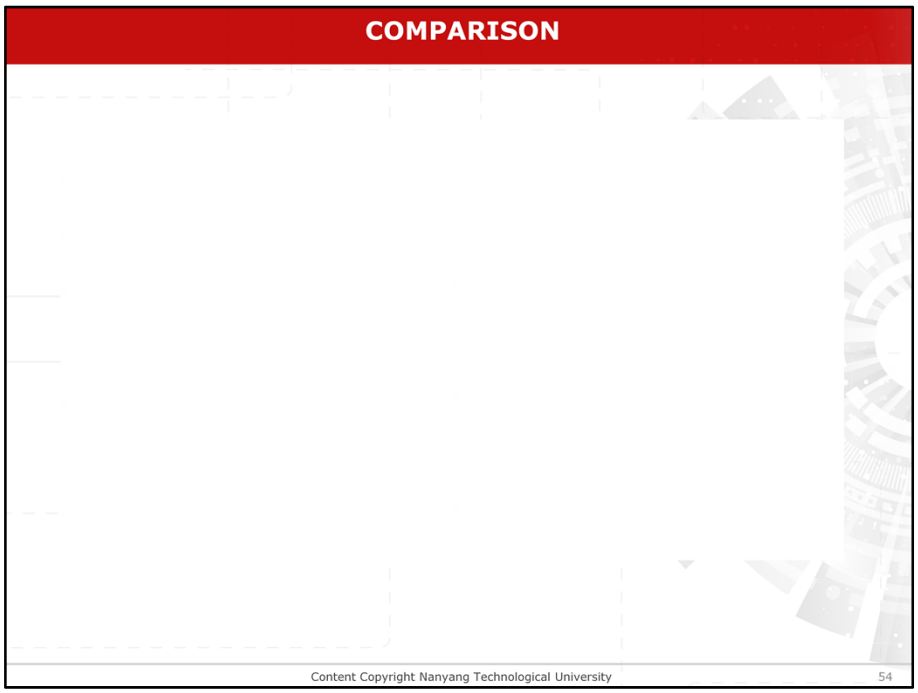


In the example program, it aims to implement a menu-driven application.



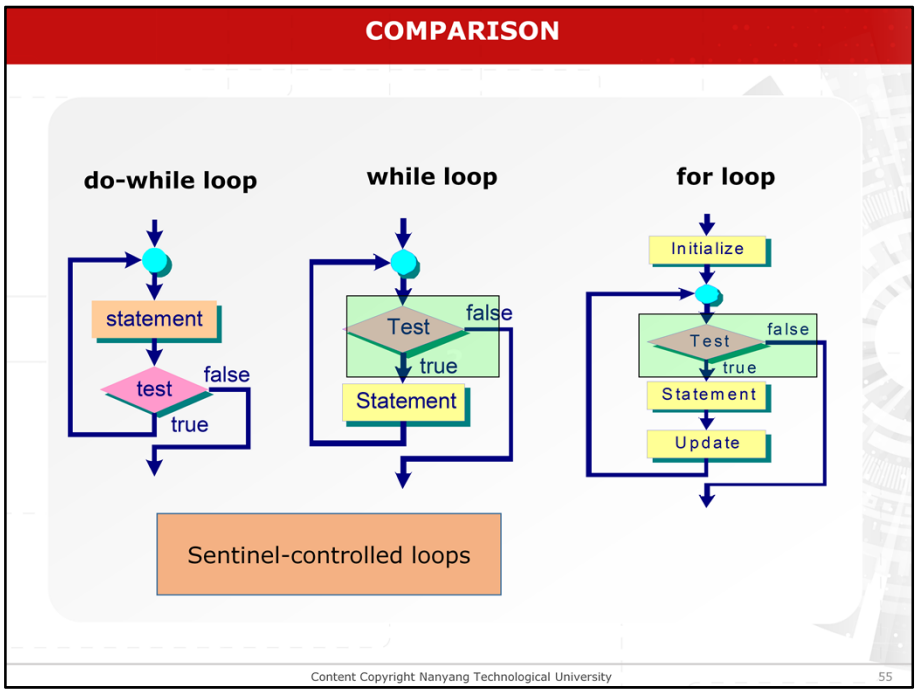
The do-while Loop: Example

The program reads in a number between 1 and 5. If the number entered is not within the range, an error message is printed on the display to prompt the user to input the number again. The program will read the user input at least once.

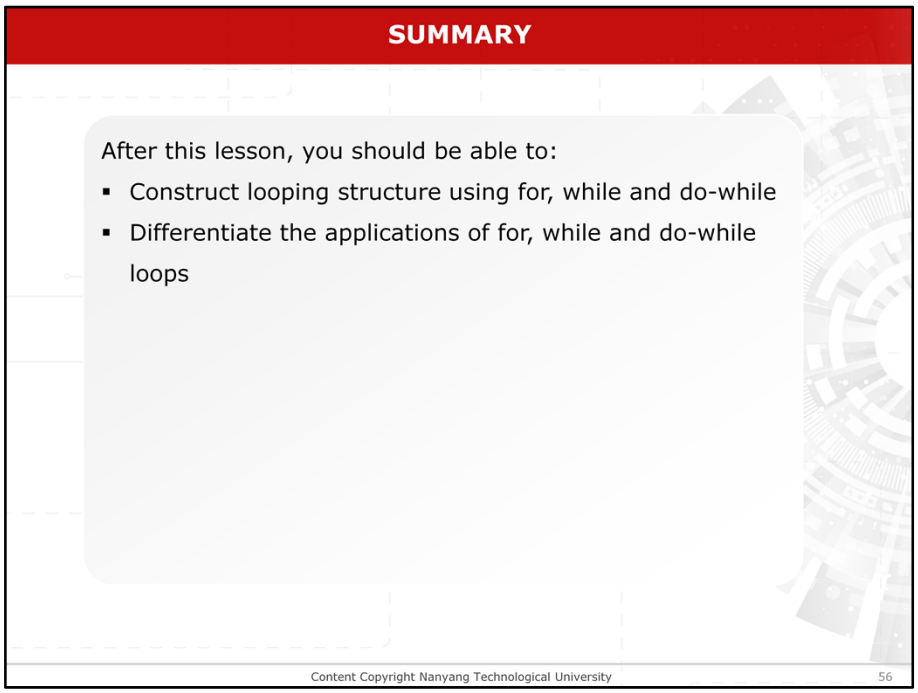


Comparison

The **do-while** loop differs from the **while** loop in that the **while** loop evaluates the **test** condition at the beginning of the loop, whereas the **do-while** loop evaluates the **test** condition at the end of the loop. If the initial **test** condition is true, the two loops will have the same number of iterations. The number of iterations between the two loops will differ only when the initial **test** condition is false. In this case, the **while** loop will exit without executing any statements in the loop body. But the **do-while** loop will execute the loop body at least once before exiting from the loop. Therefore, the **do-while** statement is useful for situations where it requires executing the loop body at least once.



Both the **while** loop and **do-while** loop can be used as sentinel-controlled loops. They can be used in situations where we do not know the number of iterations in advance. When it comes to choosing which looping statement to use, the **while** loop is generally preferred as it provides the extra control even on executing the loop body for the first time.



In summary, after viewing this video lesson, your should be able to do the listed.