

This lesson is on control structures looping

## OVERVIEW

The following are the coverage for Control structures -

### Looping

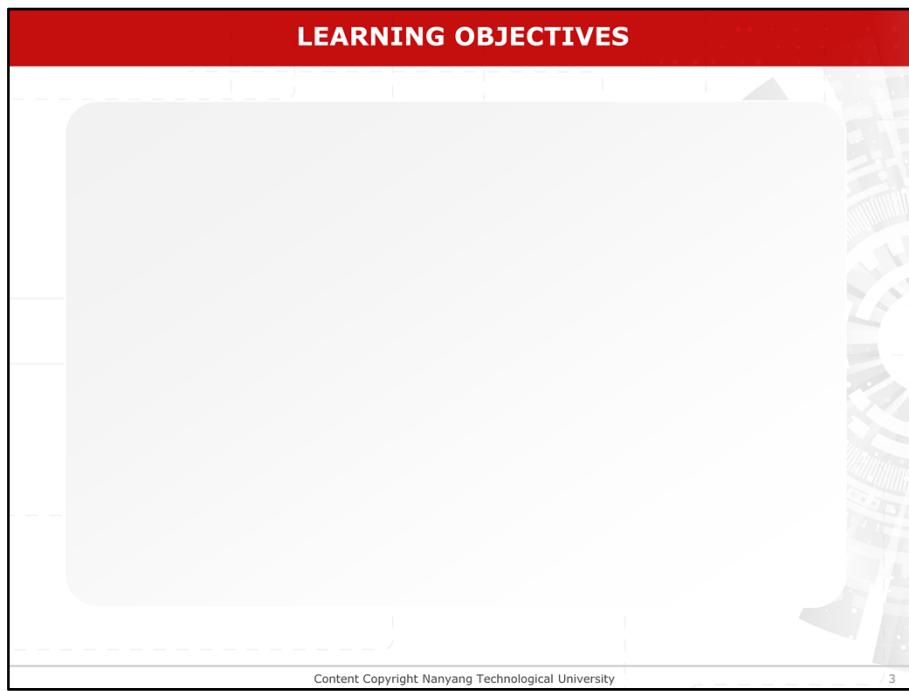
- Examples of Looping Structures
- The break and continue Statements
- Nested Loops

Content Copyright Nanyang Technological University

2

## Basic C Programming

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



Learrning objectives

**LEARNING OBJECTIVES**

At this lesson, you should be able to:

Content Copyright Nanyang Technological University 4

At this lesson, you should be able to:

## LEARNING OBJECTIVES

At this lesson, you should be able to:

- Construct looping structure using for, while and do-while

Content Copyright Nanyang Technological University 5

Construct looping structure using for, while and do-while

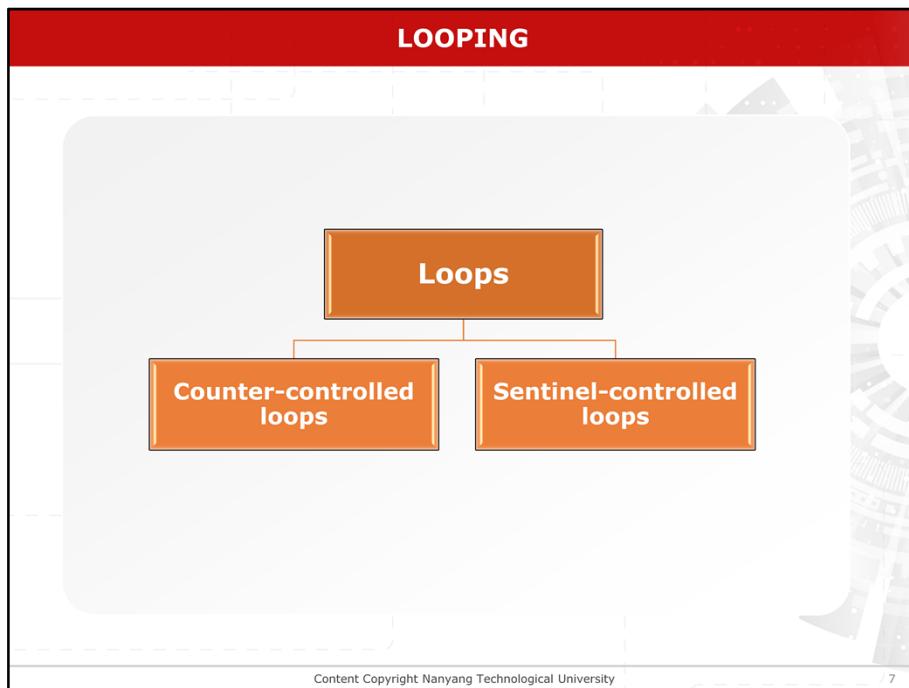
## LEARNING OBJECTIVES

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

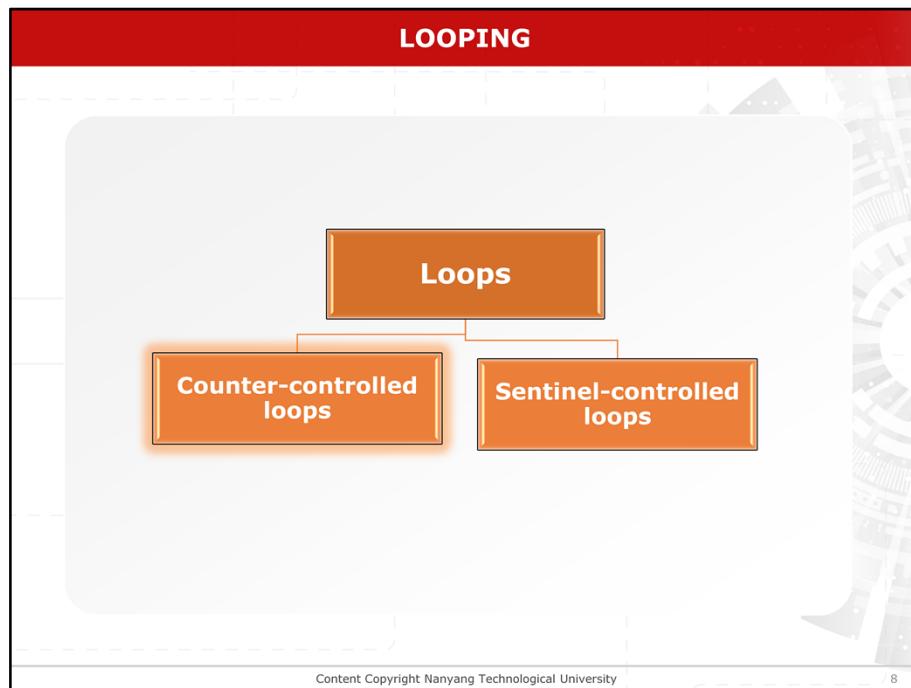
Content Copyright Nanyang Technological University 6

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

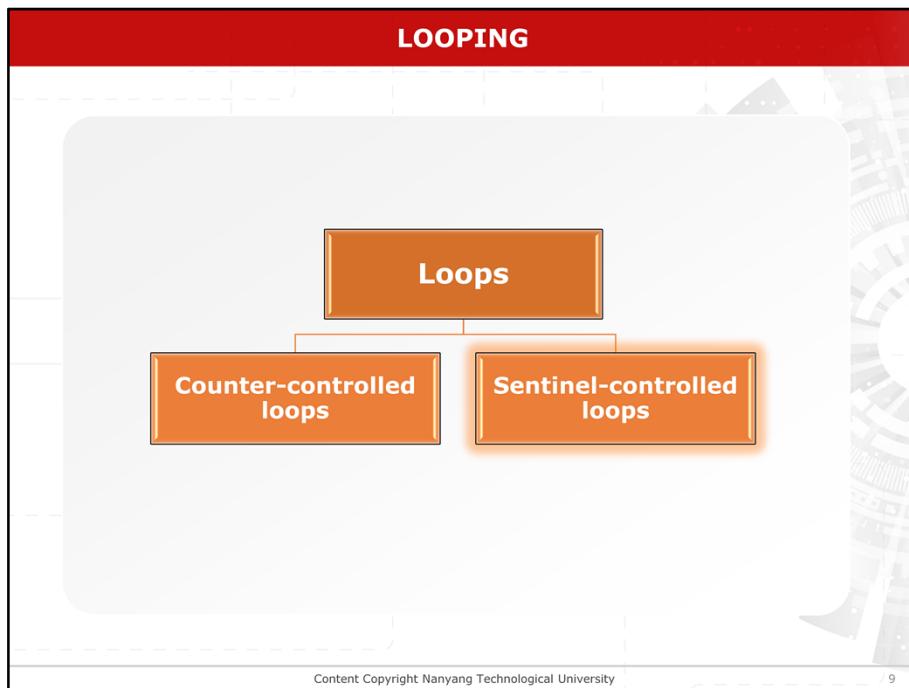


### Looping

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.

## LOOPING

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

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

## LOOPING

- 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.

Content Copyright Nanyang Technological University

11

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.

## LOOPING

- 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.

Content Copyright Nanyang Technological University

12

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.

## STEPS TO CONSTRUCT LOOPS

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

Content Copyright Nanyang Technological University

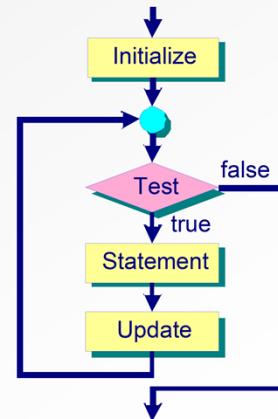
13

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

## STEPS TO CONSTRUCT LOOPS

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

- **Initialize** – initialize the loop control variable.



Content Copyright Nanyang Technological University

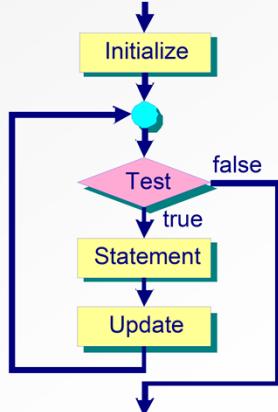
14

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

## STEPS TO CONSTRUCT LOOPS

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

- **Initialize** – initialize the [loop control variable](#).
- **Test condition** – evaluate the test condition (involve [loop control variable](#)).



```

graph TD
    Initialize[Initialize] --> Test{Test}
    Test -- true --> Statement[Statement]
    Statement --> Update[Update]
    Update --> Test
    Test -- false --> End(( ))
  
```

The flowchart illustrates the four steps: Initialize, Test, Statement, and Update. It starts with 'Initialize' leading to a decision diamond 'Test'. If the 'Test' is 'true', it leads to 'Statement', then 'Update', and loops back to 'Test'. If 'Test' is 'false', it leads directly to the end of the loop.

Content Copyright Nanyang Technological University

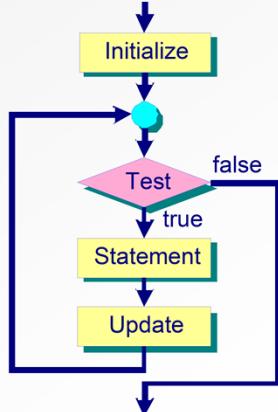
15

**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.

## STEPS TO CONSTRUCT LOOPS

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

- **Initialize** – initialize the [loop control variable](#).
- **Test condition** – evaluate the test condition (involve [loop control variable](#)).
- **Loop body** – the loop body is executed if test is true.



```

graph TD
    Initialize[Initialize] --> Test{Test}
    Test -- true --> Statement[Statement]
    Statement --> Update[Update]
    Update --> Test
    Test -- false --> End(( ))
  
```

The flowchart illustrates the four steps: Initialize, Test, Statement, and Update. It starts with 'Initialize' leading to 'Test'. 'Test' has two paths: 'true' leading to 'Statement', and 'false' leading to an end node. 'Statement' leads to 'Update', which then loops back to 'Test'.

Content Copyright Nanyang Technological University

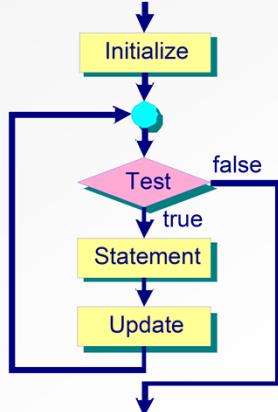
16

**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.

## STEPS TO CONSTRUCT LOOPS

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

- **Initialize** – initialize the [loop control variable](#).
- **Test condition** – evaluate the test condition (involve [loop control variable](#)).
- **Loop body** – the loop body is executed if test is true.
- **Update** – typically, [loop control variable](#) is [modified](#) through the execution of the loop body. It can then go through the **test** condition.



```

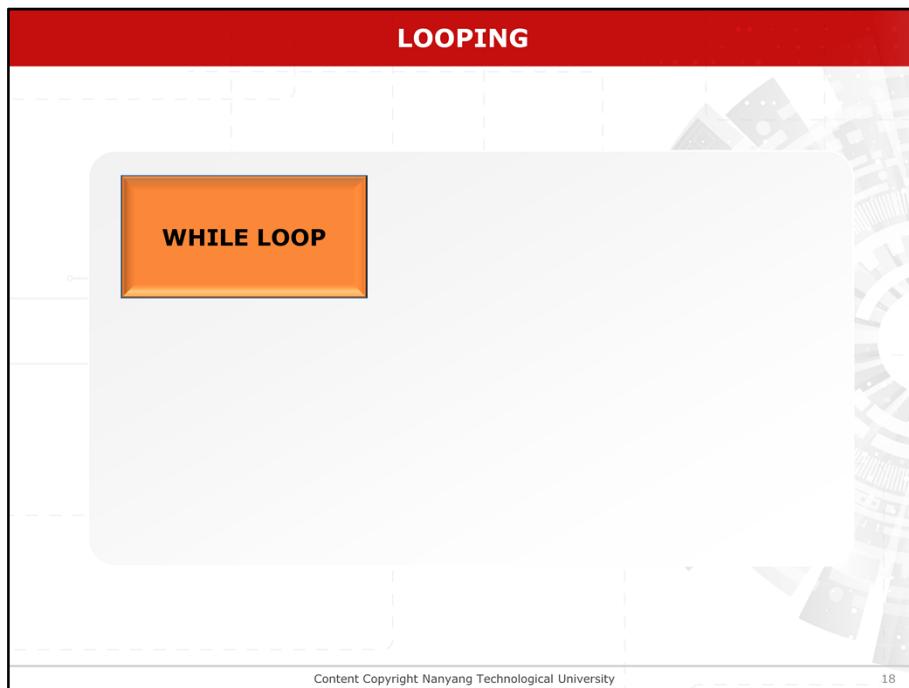
graph TD
    Initialize[Initialize] --> Test{Test}
    Test -- true --> Statement[Statement]
    Statement --> Update[Update]
    Update --> Test
    Test -- false --> End(( ))
  
```

The flowchart illustrates the iterative process of constructing a loop. It begins with the 'Initialize' step, followed by the 'Test' step (a diamond). If the test condition is 'true', it proceeds to the 'Statement' step (a rectangle), then to the 'Update' step (another rectangle). After the 'Update' step, the flow returns to the 'Test' step to re-evaluate the condition. If the test condition is 'false', the loop exits and proceeds to the next step.

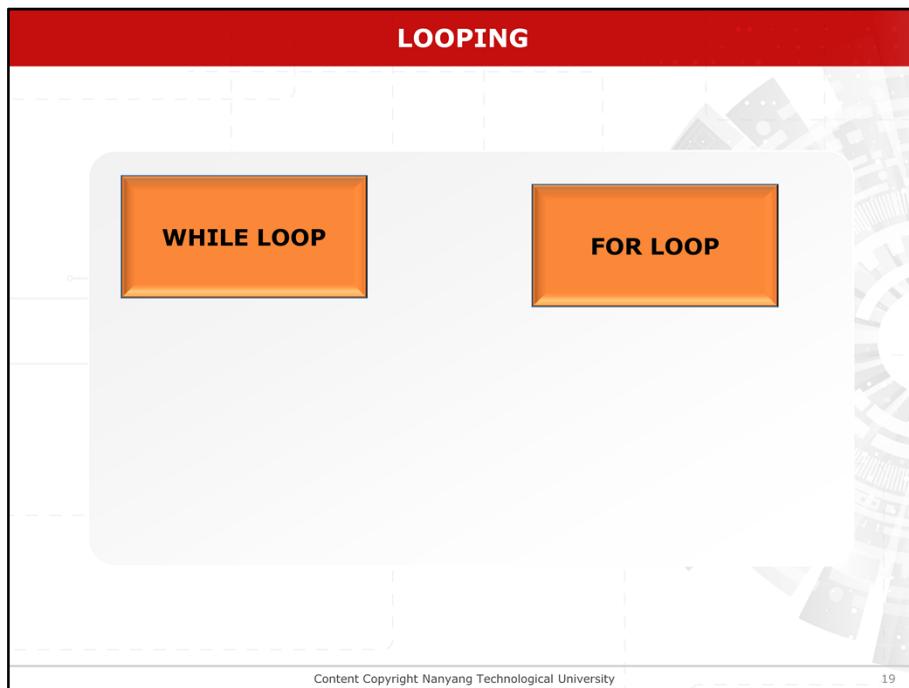
Content Copyright Nanyang Technological University

17

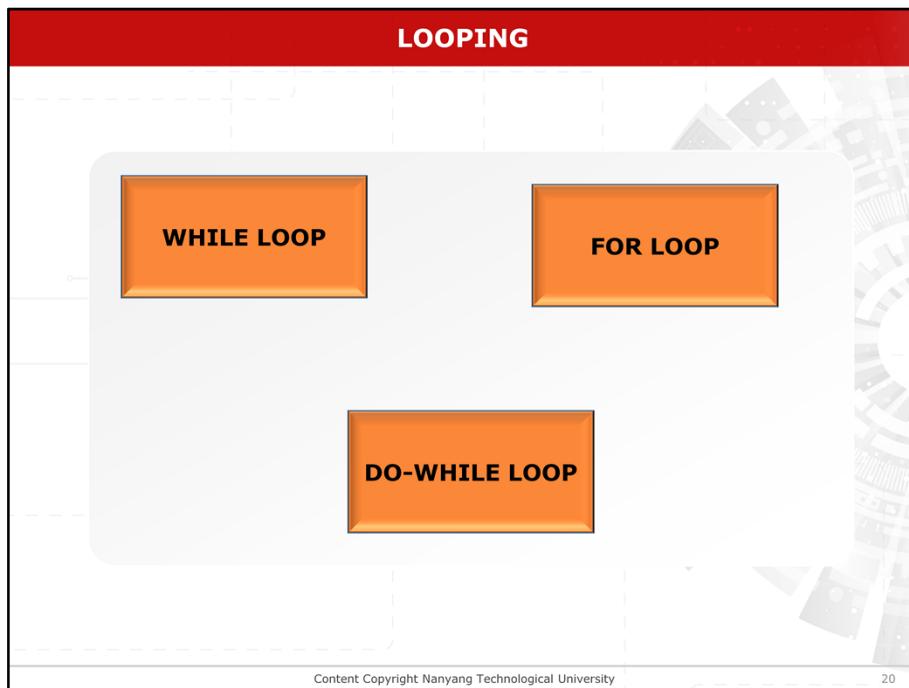
**Update** - The **test** condition typically involves the loop control variable that should be modified each time through the execution of the loop body. The loop control variable will go through the **test** condition again to determine whether to repeat the loop body again.



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



the **for** loop



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

## LOOPING

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

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

## LOOPING

- 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.

The diagram illustrates the structure of a while loop. It begins with a blue oval at the top, followed by a pink diamond labeled 'Test'. A green arrow labeled 'True' points from the 'Test' diamond down to a yellow rectangle labeled 'Statement'. Another green arrow labeled 'False' points from the 'Test' diamond to the right, leading to a blue arrow that loops back up to the blue oval at the top.

Content Copyright Nanyang Technological University 22

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

## LOOPING

- 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.

Content Copyright Nanyang Technological University

23

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

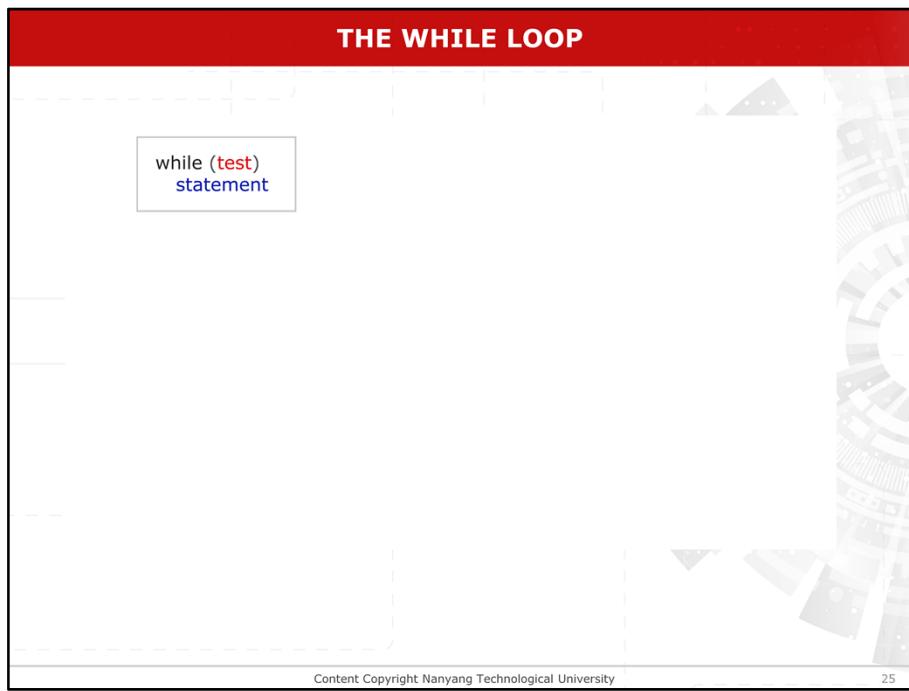
## LOOPING

- 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.

Content Copyright Nanyang Technological University

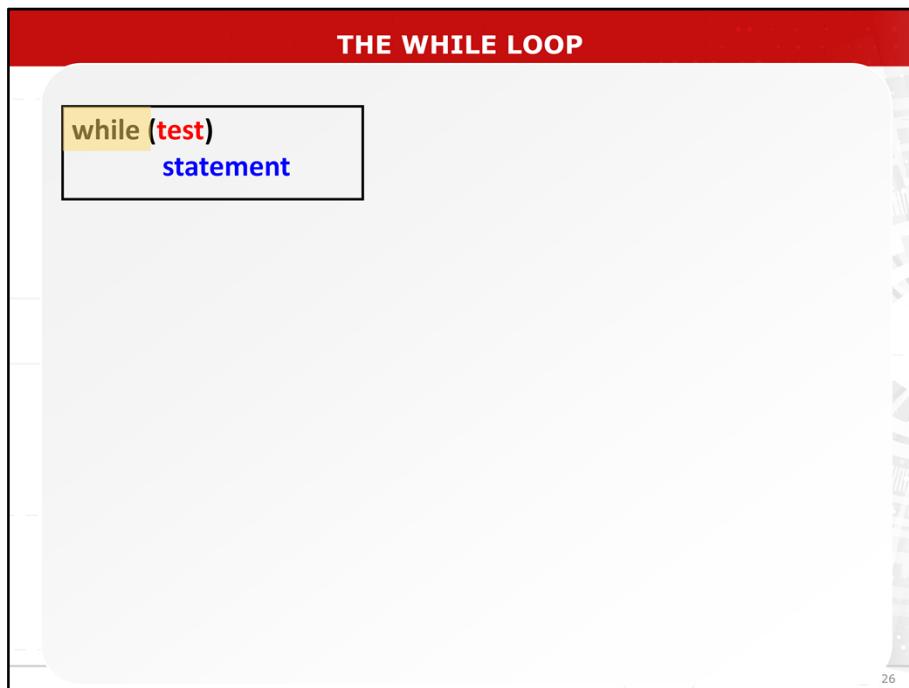
24

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

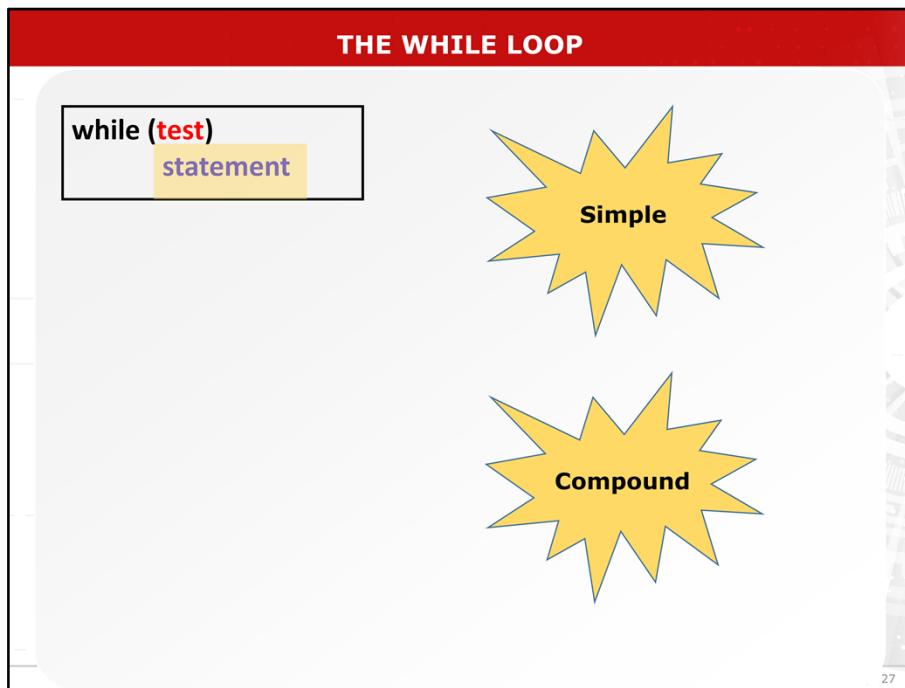


### The while Loop

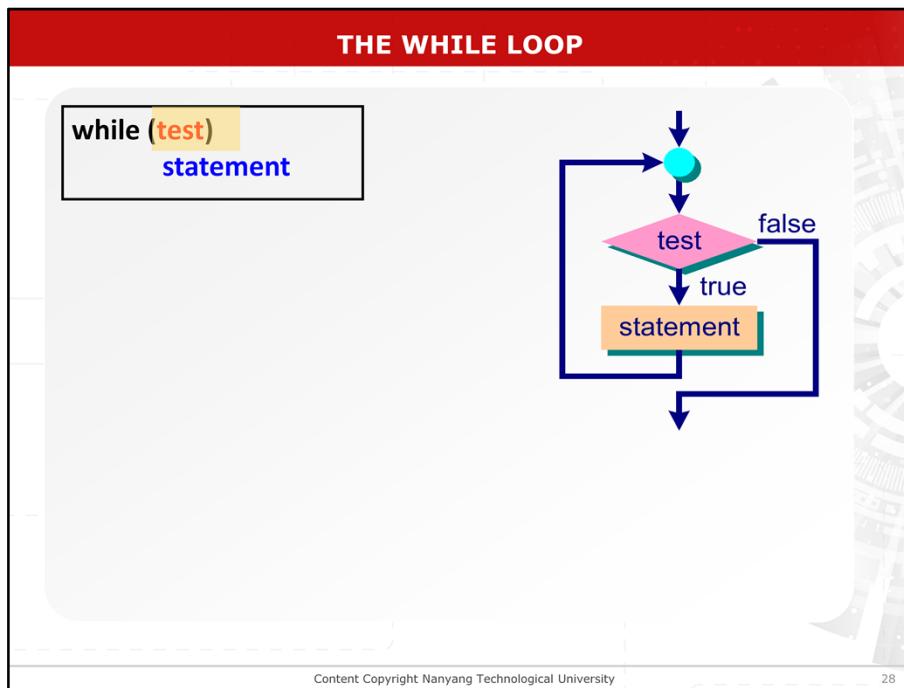
The format of the **while** statement is shown here.



where **while** is a reserved keyword.

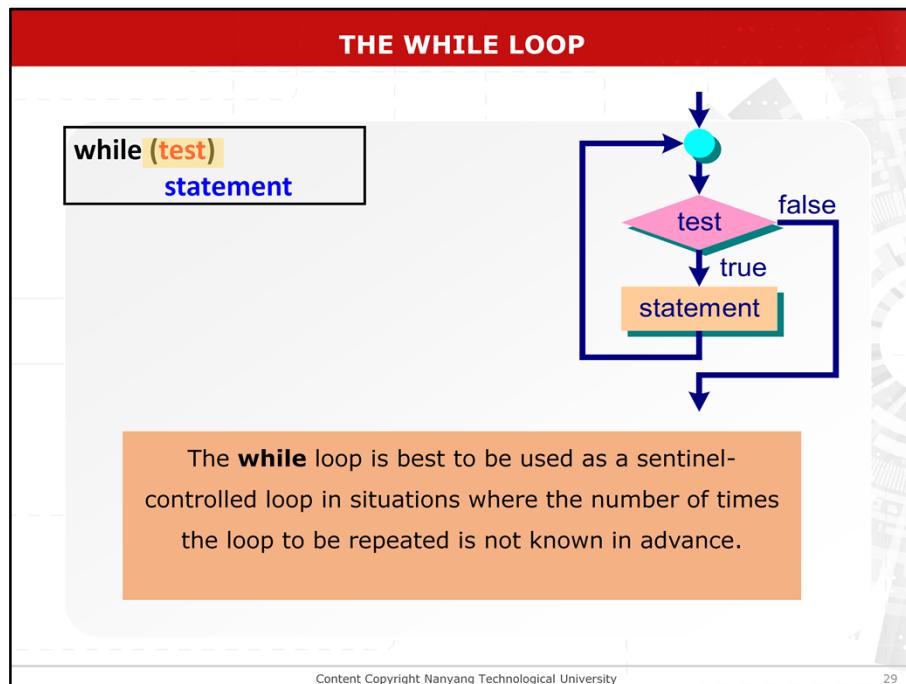


**statement** can be a simple statement terminated by a semicolon or a compound statement enclosed by **braces**.



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.

## THE WHILE LOOP

**while (*test*)  
statement**

**Sentinel-controlled Loop**

```
/* sum up a list of integers.  
The list of integers is terminated by -1. */  
#include <stdio.h>  
int main()  
{  
    int sum=0, item;  
    printf("Enter the list of integers:\n");  
  
    /* write while loop code here */  
  
    printf("The sum is %d\n", sum);  
    return 0;  
}
```

Content Copyright Nanyang Technological University

30

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

## THE WHILE LOOP

**while (*test*)  
statement**

```
/* sum up a list of integers.
The list of integers is terminated by -1. */
#include <stdio.h>
int main()
{
    int sum=0, item;
    printf("Enter the list of integers:\n");

    scanf("%d", &item);
    while (item != -1) {
        sum += item;
        scanf("%d", &item);
    }
    printf("The sum is %d\n", sum);
    return 0;
}
```

**Sentinel-controlled Loop**

**Output**  
Enter the list of integers:  
18 11 24 36 48 67 -1  
The sum is 195

Enter the list of integers:  
-1  
The sum is 0

Content Copyright Nanyang Technological University

31

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 slide has a red header bar with the title "THE FOR LOOP". Below the header, there is a code box containing the syntax: `for (initialize; test; update)  
 statement;`. To the right of the code box, the text "Counter-controlled Loop" is written. At the bottom of the slide, there is a footer with the text "Content Copyright Nanyang Technological University" and the number "32".

### 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.

THE FOR LOOP

Counter-controlled Loop

```
for (initialize; test; update)  
    statement;
```

Content Copyright Nanyang Technological University

33

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 LOOP

The **for** statement can be represented by using a **while** statement as follows:

```
initialize;  
while (test) {  
    statement;  
    update;  
}
```

Content Copyright Nanyang Technological University

34

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

## THE FOR LOOP

The **for** statement can be represented by using a **while** statement as follows:

```
initialize;  
while (test) {  
    statement;  
    update;  
}
```

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.

Content Copyright Nanyang Technological University

35

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

The **for** loop is mainly used as a counter-controlled loop.

```
for (n=0; n<10; ++n)
{
    sum += 5;
}
```

It will 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.

Content Copyright Nanyang Technological University

36

The **for** loop is mainly used as a counter-controlled loop. For example, the code shown here will 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.

## THE FOR LOOP

Consider the statements:

```
for (n=100; n>10; n-=5 )  
{  
    total += 5;  
}
```

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.

Content Copyright Nanyang Technological University 37

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.

## THE FOR LOOP

Any or all of the 3 expressions may be omitted in the **for** loop.

```
for (n=5; n<10 && n>=1; )
{
    scanf("%d", &n);
}
```

The statements shown here are valid and the **update** expression is omitted.

Content Copyright Nanyang Technological University

38

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.

## THE FOR LOOP

Notice that complex test conditions can be set using relational operators.

```
for (; n<=10; ++n)
{
    statement;
    ...
}
```

The statements shown here are valid when the **initialize** expression is omitted.

Content Copyright Nanyang Technological University

39

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

## THE FOR LOOP

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.

```
for (;;) /*an infinite loop*/  
{  
    statement;  
    ....  
}
```

The semicolons must be included even if the expression is omitted.

Content Copyright Nanyang Technological University

40

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.

## THE FOR LOOP

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

```
for(count=0, sum=0; count<5; count++)  
{  
    sum+=count;  
}
```

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

Content Copyright Nanyang Technological University

41

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**.

## THE FOR LOOP

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

```
for (count=0, sum=0; count<5; count++)  
{  
    sum+=count;  
}
```

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

```
for (count=0, sum=0; count<5; sum+=count, count++);
```

Content Copyright Nanyang Technological University

42

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

```

graph TD
    Initialize --> Start(( ))
    Start --> Test{Test}
    Test -- True --> Statement[Statement]
    Statement --> Update[Update]
    Update --> Test
    Test -- False --> End(( ))
  
```

**for (initialize; test; update)  
statement;**

```

/* display the distance a body falls in feet/sec
for the first n seconds, n input by the user */
#include <stdio.h>
#define ACCELERATION 32.0
int main()
{
    int timeLimit, t;
    int distance; /* Distance by the falling body. */
    printf("Enter the time limit(seconds):");
    scanf("%d", &timeLimit);
    for (t = 1; t <= timeLimit; t++) {
        distance = 0.5 * ACCELERATION * t * t;
        printf("Dist after %d seconds is %d \
feet.\n", t, distance);
    }
    return 0;
}
  
```

Content Copyright Nanyang Technological University 43

### 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**

<b>for (initialize; test; update)</b> <b>    statement;</b> <pre style="font-family: monospace; margin-top: 10px;"> /* display the distance a body falls in feet/sec for the first n seconds, n input by the user */ #include &lt;stdio.h&gt; #define ACCELERATION 32.0 int main() {     int timeLimit, t;     int distance; /* Distance by the falling body. */     printf("Enter the time limit(seconds):");     scanf("%d", &amp;timeLimit);     for (<b>t = 1; t &lt;= timeLimit; t++</b>) {         distance = 0.5 * ACCELERATION * t * t;         printf("Dist after %d seconds is %d \               feet.\n", t, distance);     }     return 0; }</pre>	<b>Counter-controlled Loop</b> <b>Output</b> Enter the time limit(seconds): <b>5</b> Dist after 1 seconds is 16 feet. Dist after 2 seconds is 64 feet. Dist after 3 seconds is 144 feet. Dist after 4 seconds is 256 feet. Dist after 5 seconds is 400 feet.  Enter the time limit(seconds): <b>0</b>
--	--

Content Copyright Nanyang Technological University

44

### 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**.

**THE FOR LOOP: EXAMPLE**

<pre><b>for (initialize; test; update)</b>     <b>statement;</b></pre> <p style="margin-top: 10px;"><b>/* To calculate a series of data</b></p> <div style="border: 2px solid red; padding: 5px; background-color: #f0f0f0;"> <math display="block">1 - (x/1!) + (x^2/2!) - (x^3/3!) + (x^4/4!) - \dots + (x^{20}/20!)</math> </div> <pre>#include &lt;stdio.h&gt; int main() {     double x, result = 1.0, nom = 1.0;     int n, sign=1, denom = 1;     printf("Please enter the value of x: ");     scanf("%lf", &amp;x);     for (<b>n=1; n&lt;=20; n++</b>) {         denom *= n;         sign = -sign;         nom *= x;         result += sign * nom/denom;     }     printf("The result is %lf\n", result);     return 0; }</pre>	<b>Counter-controlled Loop</b>
--	--------------------------------

Content Copyright Nanyang Technological University

45

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.

**THE FOR LOOP: EXAMPLE**

```
for (initialize; test; update)
    statement;
```

```
/* To calculate a series of data
1 - (x/1!) + (x^2/2!) - (x^3/3!) + (x^4/4!) - ...
+ (x^20/20!) */

#include <stdio.h>
int main()
{
    double x, result = 1.0, nom = 1.0;
    int n, sign=1, denom = 1;
    printf("Please enter the value of x: ");
    scanf("%lf", &x);
    for (n=1; n<=20; n++) {
        denom *= n;
        sign = -sign;
        nom *= x;
        result += sign * nom/denom;
    }
    printf("The result is %lf\n", result);
    return 0;
}
```

**Counter-controlled Loop**

Remove Slide

Content Copyright Nanyang Technological University

46

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.

**THE FOR LOOP: EXAMPLE**

<pre><b>for (initialize; test; update)</b>     <b>statement;</b></pre> <p style="color: blue; margin-top: 10px;"><i>/* To calculate a series of data  <math>1 - (x/1!) + (x^2/2!) - (x^3/3!) + (x^4/4!) - \dots</math>  <math>+ (x^{20}/20!)</math> */</i></p> <pre>#include &lt;stdio.h&gt; int main() {     double x, result = 1.0; nom = 1.0;     int n, sign=1, denom = 1;     printf("Please enter the value of x: ");     scanf("%lf", &amp;x);     for (n=1; n&lt;=20; n++) {         denom *= n;         sign = -sign;         nom *= x;         result += sign * nom/denom;     }     printf("The result is %lf\n", result);     return 0; }</pre>	<b>Counter-controlled Loop</b>
---	--------------------------------

Remove Slide

Content Copyright Nanyang Technological University

47

- . 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 FOR LOOP: EXAMPLE

```
for (initialize; test; update)
    statement;

/* To calculate a series of data
1 - (x/1!) + (x^2/2!) - (x^3/3!) + (x^4/4!) - ...
+ (x^20/20!) */

#include <stdio.h>
int main()
{
    double x, result = 1.0, nom = 1.0;
    int n, sign=1, denom = 1;
    printf("Please enter the value of x: ");
    scanf("%lf", &x);

    for (n=1; n<=20; n++) {
        denom *= n;
        sign = -sign;
        nom *= x;
        result += sign * nom/denom;
    }

    printf("The result is %lf\n", result);
    return 0;
}
```

Remove Slide

Content Copyright Nanyang Technological University

48

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

## THE FOR LOOP: EXAMPLE

```
for (initialize; test; update)
    statement;

/* To calculate a series of data
1 - (x/1!) + (x^2/2!) - (x^3/3!) + (x^4/4!) - ...
+ (x^20/20!) */

#include <stdio.h>
int main()
{
    double x, result = 1.0, nom = 1.0;
    int n, sign=1, denom = 1;
    printf("Please enter the value of x: ");
    scanf("%lf", &x);

    for (n=1; n<=20; n++) {
        denom *= n;
        sign = -sign;
        nom *= x;
        result += sign * nom/denom;
    }

    printf("The result is %lf\n", result);
    return 0;
}
```

Remove Slide

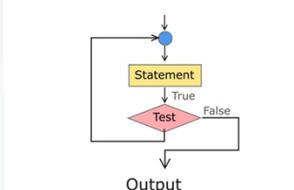
Content Copyright Nanyang Technological University

49

and prints the result.

## THE DO-WHILE LOOP

```
do  
    statement;  
while (test);
```



C also provides the **do-while** statement which implements a control pattern different from the **while** and **for** loops.

Content Copyright Nanyang Technological University

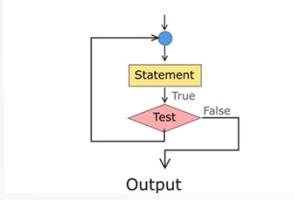
50

### 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

```
do  
    statement;  
while (test);
```



```
graph TD; Start(( )) --> Statement[Statement]; Statement --> Test{Test}; Test -- True --> Statement; Test -- False --> Output[Output];
```

The flowchart illustrates the execution of a do-while loop. It begins with an entry point at the top, leading to a rectangular box labeled "Statement". An arrow then points to a diamond-shaped box labeled "Test". If the condition in "Test" is "True", the flow returns to the "Statement" box. If the condition is "False", the flow proceeds directly to the "Output" box at the bottom.

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.

Content Copyright Nanyang Technological University

51

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.

Dr Hui Siu Cheung, SCSE, NTU

51

## THE DO-WHILE LOOP

```
do
    statement;
while (test);

/* Menu-Based User Selection */
#include <stdio.h>
int main()
{
    int input; /* User input number. */
    do {
        /* display menu */
        printf("Input a number >= 1 and <= 5: ");
        scanf("%d",&input);
        if (input > 5 || input < 1)
            printf("%d is out of range.\n", input);
    } while (input > 5 || input < 1);
    printf("Input = %d\n", input);
    return 0;
}
```

Output

Content Copyright Nanyang Technological University 52

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

**THE DO-WHILE LOOP: EXAMPLE**

```
do
    statement;
while (test);
```

```
/* Menu-Based User Selection */
#include <stdio.h>
int main()
{
    Int input; /* User input number. */
    do {
        /* display menu */
        printf("Input a number >= 1 and <= 5: ");
        scanf("%d",&input);
        if (input > 5 || input < 1)
            printf("%d is out of range.\n", input);
    } while (input > 5 || input < 1);
    printf("input = %d\n", input);
    return 0;
}
```

**for Menu-driven Applications**

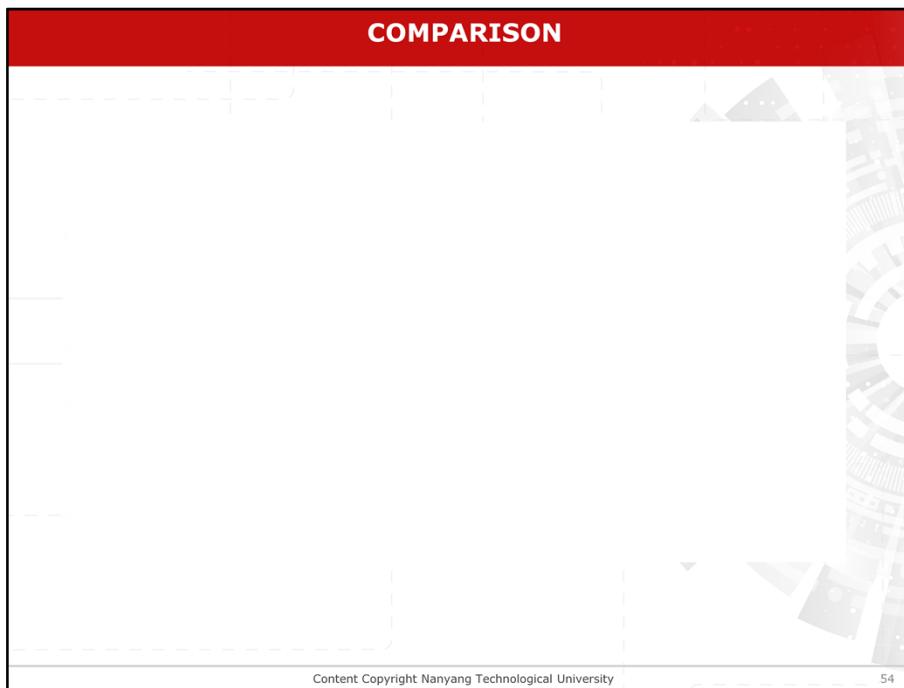
Content Copyright Nanyang Technological University

**Output**

Input a number >= 1 and <= 5: 6  
6 is out of range.  
Input a number >= 1 and <= 5: 5  
Input = 5

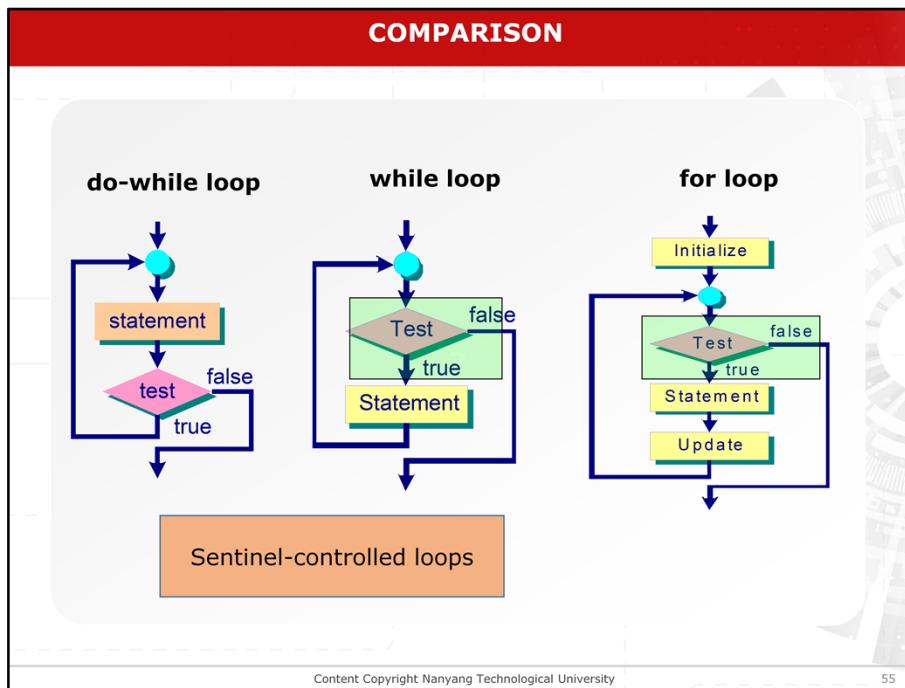
### The do-while Loop: Example

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.

The slide has a red header bar with the word "SUMMARY" in white capital letters. Below the header is a large white rectangular area with rounded corners. Inside this area, there is a smaller white box containing the following text:  
After 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

At the bottom left of the white area, it says "Content Copyright Nanyang Technological University". At the bottom right, it says "56". The background of the slide features a faint watermark of a circular emblem or seal.

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