

1. Learning Objectives

By the end of this lecture, students will be able to:

- Understand the purpose of the `do-while` loop.
- Identify its syntax and working.
- Explain the difference between `while` and `do-while`.
- Write programs using `do-while`.
- Modify existing programs and solve coding exercises.

2. What Is a `do-while` Loop?

A `do-while` loop repeats a block of code while a condition is true.

The important difference is that the `do-while` loop executes its body at least once, even if the condition is false at the beginning.

Real-Life Example

Imagine a university ATM menu. When a user inserts their card, the ATM displays a menu. After performing an operation, the user may choose to display the menu again.

The menu must appear at least once before the program can ask whether the user wants to continue.

3. Syntax of the `do-while` Loop

```
do
{
    // Statements to execute
}
while (condition);
```

Important: Notice the semicolon `;` after `while (condition)`.

The execution order is:

1. Execute the loop body.
2. Check the condition.
3. If true, repeat the loop body.
4. If false, stop the loop.

4. Example 1: Display a Message Five Times

```
#include <iostream>
using namespace std;

int main()
{
    int i = 1;
```

```
do
{
    cout << "Welcome to Ghazi University!" << endl;
    i++;
}
while (i <= 5);

return 0;
}
```

Understand the Code

- `int i = 1;` initializes the counter.
- `do` marks the beginning of the loop body.
- `cout` displays the message.
- `i++` increases the counter by 1.
- `while (i <= 5);` checks whether the loop should repeat.

Output:

```
Welcome to Ghazi University!
Welcome to Ghazi University!
Welcome to Ghazi University!
Welcome to Ghazi University!
Welcome to Ghazi University!
```

Modify the Code

1. Display the message 10 times.
2. Change the message to `"I am learning C++."`.
3. Start the counter at 3 and change the condition to `i <= 7`.
4. Display the value of `i` along with the message.

5. Example 2: Print Numbers from 1 to 10

```
#include <iostream>
using namespace std;

int main()
{
    int i = 1;

    do
    {
        cout << i << " ";
        i++;
    }
    while (i <= 10);
}
```

```
    return 0;
}
```

Output:

```
1 2 3 4 5 6 7 8 9 10
```

Understand the Code

The counter starts at 1. The loop prints the current value and increases it by 1. The condition is checked after each repetition. When `i` becomes 11, the condition is false and the loop stops.

Modify the Code

1. Print numbers from 1 to 20.
2. Print numbers from 10 down to 1.
3. Print even numbers from 2 to 20.
4. Print odd numbers from 1 to 19.

Hint for descending numbers: start from 10, use `i >= 1`, and decrease the counter using `i--`.

6. Example 3: Execute the Loop at Least Once

This is the most important example for understanding `do-while`.

```
#include <iostream>
using namespace std;

int main()
{
    int i = 10;

    do
    {
        cout << "Value of i = " << i << endl;
        i++;
    }
    while (i <= 5);

    return 0;
}
```

Output:

```
Value of i = 10
```

Understand the Code

- The counter starts at 10.
- The loop body executes and prints 10.
- The counter becomes 11.
- The condition `i <= 5` is false.
- The loop stops.

Although the condition was false, the message was displayed once because the condition is checked after executing the loop body.

Modify the Code

1. Change `i = 10` to `i = 3`. Predict the output before running the program.
2. Change the condition to `i <= 12`.
3. Remove `i++` and explain what happens when the condition is true.

7. Example 4: Take Input from the User

Write a program that asks the user to enter a number and then displays the numbers from 1 up to that number.

```
#include <iostream>
using namespace std;

int main()
{
    int n;
    int i = 1;

    cout << "Enter a number: ";
    cin >> n;

    do
    {
        cout << i << " ";
        i++;
    }
    while (i <= n);

    return 0;
}
```

Sample output:

```
Enter a number: 5
1 2 3 4 5
```

Understand the Code

If the user enters 5, the program displays numbers from 1 to 5.

However, if the user enters 0, the program still displays 1 once. Why? Because **do-while** executes the loop body before checking the condition.

Modify the Code

1. Ask the user for a number and display a countdown from that number to 1.
2. Ask the user for a number and display its multiplication table from 1 to 10.
3. Ask the user for a number and calculate the sum from 1 to that number.

8. Example 5: Calculate the Sum of Numbers

```
#include <iostream>
using namespace std;

int main()
{
    int i = 1;
    int sum = 0;

    do
    {
        sum = sum + i;
        i++;
    }
    while (i <= 5);

    cout << "Sum = " << sum;

    return 0;
}
```

Output:

```
Sum = 15
```

Understand the Code

- **i** controls the loop.
- **sum** stores the running total.
- **sum = sum + i;** adds the current number to the total.
- **i++** moves to the next number.
- The loop stops when **i** becomes 6.

Current i	Calculation	sum
1	0 + 1	1
2	1 + 2	3
3	3 + 3	6

Current i	Calculation	sum
4	6 + 4	10
5	10 + 5	15

Modify the Code

1. Calculate the sum from 1 to 10.
2. Calculate the sum from 1 to 100.
3. Calculate the sum of even numbers from 2 to 20.

9. Example 6: Menu-Driven Program

A menu-driven program allows a user to select an option. The menu is displayed again until the user chooses to exit.

```
#include <iostream>
using namespace std;

int main()
{
    int choice;

    do
    {
        cout << "\n--- University Menu ---" << endl;
        cout << "1. Student Information" << endl;
        cout << "2. Display Message" << endl;
        cout << "3. Exit" << endl;

        cout << "Enter your choice: ";
        cin >> choice;

        if (choice == 1)
        {
            cout << "Student Information Selected" << endl;
        }
        else if (choice == 2)
        {
            cout << "Welcome to Ghazi University!" << endl;
        }
        else if (choice == 3)
        {
            cout << "Program ended." << endl;
        }
        else
        {
            cout << "Invalid choice!" << endl;
        }
    }
}
```

```

    while (choice != 3);

    return 0;
}

```

Understand the Code

- The menu appears when the loop begins.
- The user enters a choice.
- **if-else-if** determines which message to display.
- If the user enters 1 or 2, the menu appears again.
- If the user enters 3, the condition **choice != 3** becomes false, and the loop stops.

Modify the Code

1. Add option 4: Display "Admissions Fall 2026".
2. Add an option to display the university name.
3. Change the exit option from 3 to 5. Update all relevant conditions.
4. Add a message for invalid choices.

10. Example 7: Loop Runs At Least Once (Important Concept)

```

#include <iostream>
using namespace std;

int main() {
    int i = 10;

    do {
        cout << "This will print once\n";
    } while (i < 5);

    return 0;
}

```

Explanation:

- Condition is **false**
- Still, the message prints **once**
- This is the main difference from **while** loop

11. Difference Between **while** and **do-while**

Feature	while loop	do-while loop
Condition checked	Before the body	After the body
Minimum executions	Zero	One

Feature	while loop	do-while loop
Semicolon after condition	Not required	Required
Useful for	Condition-controlled repetition	Menus and actions that must happen at least once

Compare these programs.

Using **while**:

```
int i = 10;

while (i <= 5)
{
    cout << i;
}
```

Output: Nothing.

Using **do-while**:

```
int i = 10;

do
{
    cout << i;
}
while (i <= 5);
```

Output:

```
10
```

12. Common Mistakes

Mistake 1: Forgetting the semicolon

Incorrect:

```
do
{
    cout << "Hello";
}
while (i <= 5)
```

Correct:

```
do
{
    cout << "Hello";
}
while (i <= 5);
```

Mistake 2: Forgetting to update the counter

If the condition remains true and the counter never changes, the loop may run indefinitely.

Mistake 3: Expecting zero executions

A **do-while** loop always executes its body at least once, even when its condition is initially false.

13. Practice Coding Exercises

Solve [Coding Exercises 7–16](#) in the **do-while loop** section. Write, run, and test each C++ program to verify the output.

14. Lesson Summary

- The **do-while** loop executes the body before checking the condition.
- Its body always executes at least once.
- The condition is followed by a semicolon.
- Counter updates are important when the loop depends on a changing value.
- Menu-driven programs are a common practical use of **do-while**.

Next lecture: The **for** loop — syntax, counting, sequences, sum calculations, code modification, and coding exercises.