

1. Learning Objectives

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

- Understand the working of a **while** loop.
- Write a **while** loop using a condition.
- Print numbers in ascending and descending order.
- Calculate the sum of numbers using a loop.
- Trace a loop step by step.
- Identify and correct common loop errors.

2. What Is a **while** Loop?

A **while** loop repeats a block of code as long as a given condition is true.

Real-life example: Imagine filling a water bottle. You continue filling it while it is not full. When the bottle becomes full, you stop.

A **while** loop works in a similar way:

1. Check the condition.
2. If the condition is true, execute the loop body.
3. Update the variable.
4. Check the condition again.
5. Stop when the condition becomes false.

3. Syntax of the **while** Loop

```
while (condition)
{
    // Statements to repeat
}
```

Example:

```
int i = 1;

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

Understand the Code

- **int i = 1;** — starts the counter at 1.
- **while (i <= 5)** — repeats the loop while **i** is less than or equal to 5.
- **cout << i << endl;** — displays the current value of **i**.

- `i++`; — increases `i` by 1.

Output:

```
1
2
3
4
5
```

Remember: `i++` means `i = i + 1`.

4. How Does the Loop Work?

Let's trace the previous program.

Value of <code>i</code>	Condition <code>i <= 5</code>	Output
1	True	1
2	True	2
3	True	3
4	True	4
5	True	5
6	False	Loop stops

When `i` becomes 6, the condition is false, so the loop ends.

5. Example 1: Print a Message Five Times

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

int main()
{
    int i = 1;

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

    return 0;
}
```

Understand the code: The loop starts at 1 and repeats until `i` becomes greater than 5. The message is displayed five times.

Modify the code:

- Display the message 10 times.
- Change the message to "Welcome to C++ Programming!".
- Start `i` at 2 and repeat while `i <= 6`. How many times will the message appear?

6. Example 2: Print Numbers from 1 to 10

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

int main()
{
    int i = 1;

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

    return 0;
}
```

Output:

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

Understand the code: The counter begins at 1. Each repetition prints the current number and increases it by 1. The loop stops when the counter reaches 11.

Modify the code:

- Print numbers from 1 to 20.
- Print numbers from 5 to 15.
- Print numbers from 10 to 1.

7. Example 3: Print Even Numbers from 2 to 20

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

int main()
{
    int i = 2;

    while (i <= 20)
    {
        cout << i << " ";
    }
}
```

```
        i = i + 2;
    }

    return 0;
}
```

Output:

```
2 4 6 8 10 12 14 16 18 20
```

Understand the code: The counter starts at 2 and increases by 2 after each repetition. Therefore, only even numbers are displayed.

Modify the code:

- Print odd numbers from 1 to 19.
- Print even numbers from 10 to 30.
- Print multiples of 5 from 5 to 50.

8. Example 4: Countdown from 10 to 1

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

int main()
{
    int i = 10;

    while (i >= 1)
    {
        cout << i << " ";
        i--;
    }

    return 0;
}
```

Output:

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

Understand the code:

- The counter starts at 10.
- The condition checks whether `i >= 1`.
- `i--` decreases the counter by 1.
- The loop stops when `i` becomes 0.

Modify the code: Change the countdown to start from 20 and stop at 1.

9. Example 5: Sum of Numbers from 1 to 5

A loop can do more than display numbers. It can also calculate their sum.

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

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

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

    cout << "Sum = " << sum;

    return 0;
}
```

Output:

```
Sum = 15
```

Understand the Code

- **i** controls how many times the loop repeats.
- **sum** stores the running total.
- **sum = sum + i;** adds the current value of **i** to **sum**.
- **i++** moves to the next number.

Trace Table

i	Calculation	New sum
1	0 + 1	1
2	1 + 2	3
3	3 + 3	6
4	6 + 4	10
5	10 + 5	15

Modify the code:

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

10. Example 6: Take Input from the User

Suppose the user enters a number, and the program displays all 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;

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

    return 0;
}
```

Sample output:

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

Understand the code: The value of `n` comes from the user. If the user enters 5, the loop displays numbers from 1 to 5. If the user enters 8, it displays numbers from 1 to 8.

Modify the code: Display numbers from `n` down to 1.

Suggested title: **Types of `while` Loops: Counter-Controlled and Sentinel-Controlled**

Here is the improved version of your student notes, using simple English and clear examples.

11 Types of `while` Loops

A `while` loop can be used in different ways depending on how the repetition is controlled. Two common types are **counter-controlled loops** and **sentinel-controlled loops**.

1. Counter-Controlled Loop

A **counter-controlled loop** repeats a specific number of times. A counter variable keeps track of the number of repetitions.

Example: Display numbers from 1 to 5

Code

```
int i = 1;

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

How It Works

- **Initialization:** `int i = 1;` sets the starting value of the counter.
- **Condition:** `i <= 5` allows the loop to continue while `i` is less than or equal to 5.
- **Update:** `i++` increases the counter by 1 after each repetition.
- **Termination:** The loop stops when `i` becomes 6 because the condition becomes false.

Output

```
1
2
3
4
5
```

Remember: A counter-controlled loop is used when the number of repetitions is known in advance.

2. Sentinel-Controlled Loop

A **sentinel-controlled loop** repeats until a special value, called a **sentinel value**, is entered or encountered. This value signals the loop to stop.

Example: Accept numbers until the user enters 0

Code

```
int num;

cout << "Enter a number (0 to stop): ";
cin >> num;

while (num != 0)
{
    cout << "You entered: " << num << endl;
```

```
cout << "Enter a number (0 to stop): ";  
cin >> num;  
}
```

How It Works

- **Input:** The user enters a number.
- **Condition:** `num != 0` checks whether the entered number is not 0.
- **Repetition:** If the number is not 0, the loop displays it and asks for another number.
- **Termination:** When the user enters 0, the condition becomes false, and the loop stops.

Sample Output

```
Enter a number (0 to stop): 10  
You entered: 10  
Enter a number (0 to stop): 25  
You entered: 25  
Enter a number (0 to stop): 7  
You entered: 7  
Enter a number (0 to stop): 0
```

Remember: A sentinel-controlled loop is used when the number of repetitions is not known in advance. The loop continues until the sentinel value is entered.

3. Key Differences

Feature	Counter-Controlled Loop	Sentinel-Controlled Loop
Purpose	Repeats a specific number of times	Repeats until a special value is entered
Control	Counter variable	Sentinel value
Repetitions	Usually known in advance	Not known in advance
Stopping condition	Counter reaches its limit	Sentinel value is entered
Example	Display numbers from 1 to 10	Accept numbers until 0 is entered

12. Common Mistakes in a `while` Loop

Mistake 1: Forgetting to update the counter

Incorrect:

```
int i = 1;  
  
while (i <= 5)  
{
```

```
    cout << i << " ";  
}
```

Problem: The value of `i` never changes. The condition remains true, so the loop runs indefinitely.

Correct:

```
int i = 1;  
  
while (i <= 5)  
{  
    cout << i << " ";  
    i++;  
}
```

Mistake 2: Using the wrong condition

```
int i = 1;  
  
while (i >= 5)  
{  
    cout << i;  
    i++;  
}
```

Problem: The condition is false from the beginning because 1 is not greater than or equal to 5. The loop body never executes.

Mistake 3: Adding a semicolon after `while`

Incorrect:

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

Problem: The semicolon creates an empty loop. It may cause an infinite loop if the condition remains true.

Correct:

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

13. Practice Coding Exercises

Solve [Coding Exercises 1–6](#) in the [while loop](#) section. Write, run, and test each C++ program to verify the output.

14. Summary

- A [while](#) loop repeats statements while its condition is true.
- Initialize the counter before the loop.
- Write a suitable condition.
- Update the counter inside the loop when needed.
- Use trace tables to understand how values change.
- Always check that the loop will eventually stop.

Next lecture: The [do-while](#) loop — how it differs from the [while](#) loop and why it executes its body at least once.