1. Learning Objectives

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

- Understand the purpose and syntax of a `for` loop.
- Use a `for` loop to repeat statements.
- Print ascending and descending number sequences.
- Calculate sums and multiplication tables.
- Modify existing programs and solve coding exercises.

2. What Is a `for` Loop?

A `for` loop repeats a block of code a specified number of times or while a condition remains true.

Real-life example: A teacher wants to mark attendance for 30 students. Instead of writing the same instructions 30 times, a loop can repeat the task.

A `for` loop is especially useful when the repetition follows a clear counting pattern.

3. Syntax of the `for` Loop

```
for (initialization; condition; update)
{
    // Statements to repeat
}
```

There are three main parts:

Part	Purpose	Example
Initialization	Sets the starting value	`int i = 1`
Condition	Determines whether to continue	`i <= 5`
Update	Changes the counter	`i++`

Remember the order: Initialize → Check condition → Execute body → Update → Check again.

4. Example 1: Print Numbers from 1 to 5

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

int main()
{
    for (int i = 1; i <= 5; i++)
    {
        cout << i << endl;
    }
```

```
    return 0;
}
```

Output:

```
1
2
3
4
5
```

Understand the Code

- `int i = 1` starts the counter at 1.
- `i <= 5` allows the loop to run while `i` is 5 or less.
- `i++` increases `i` by 1 after each repetition.
- When `i` becomes 6, the condition is false and the loop stops.

Modify the Code

1. Print numbers from 1 to 10.
2. Print numbers from 1 to 20.
3. Start from 5 and print up to 15.
4. Display each number on the same line instead of separate lines.

5. Example 2: Print Numbers in Descending Order

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

int main()
{
    for (int i = 10; i >= 1; i--)
    {
        cout << i << " ";
    }

    return 0;
}
```

Output:

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

Understand the Code

- `int i = 10` starts from 10.

- `i >= 1` continues the loop until the counter becomes less than 1.
- `i--` decreases the counter by 1.

Modify the Code

1. Print numbers from 20 to 1.
2. Print numbers from 50 to 10.
3. Print numbers from 100 to 90.

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

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

int main()
{
    for (int i = 2; i <= 20; i = i + 2)
    {
        cout << i << " ";
    }

    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 each time. Therefore, only even numbers are displayed.

`i = i + 2` means the new value of `i` is its previous value plus 2.

Modify the Code

1. Print odd numbers from 1 to 19.
2. Print multiples of 5 from 5 to 50.
3. Print even numbers from 20 down to 2.

7. Example 4: Calculate the Sum from 1 to 10

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

int main()
{
    int sum = 0;
```

```
    for (int i = 1; i <= 10; i++)
    {
        sum = sum + i;
    }

    cout << "Sum = " << sum;

    return 0;
}
```

Output:

```
Sum = 55
```

Understand the Code

- `int sum = 0` initializes the total.
- The loop starts at 1 and continues up to 10.
- `sum = sum + i` adds each number to the running total.
- After the loop finishes, the program displays the final sum.

For example:

i	Calculation	sum
1	0 + 1	1
2	1 + 2	3
3	3 + 3	6
4	6 + 4	10
…	…	…
10	45 + 10	55

Modify the Code

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

8. Example 5: Multiplication Table

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

int main()
{
    int n;
```

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

    for (int i = 1; i <= 10; i++)
    {
        cout << n << " x " << i
             << " = " << n * i << endl;
    }

    return 0;
}
```

Sample output:

```
Enter a number: 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

Understand the Code

- `cin >> n` receives a number from the user.
- The loop repeats ten times.
- `n * i` calculates each multiplication result.
- `endl` moves the output to a new line.

Modify the Code

1. Display the table up to 20 instead of 10.
2. Display the multiplication table in reverse order, from 10 to 1.
3. Ask the user for both the number and the table's ending value.

9. Example 6: Take User Input and Print a Sequence

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

int main()
{
    int n;

    cout << "Enter the ending number: ";
```

```
    cin >> n;

    for (int i = 1; i <= n; i++)
    {
        cout << i << " ";
    }

    return 0;
}
```

Sample output:

```
Enter the ending number: 7
1 2 3 4 5 6 7
```

Understand the Code

The user decides how many numbers to print. If the user enters 7, the loop prints 1 through 7.

Modify the Code

1. Ask the user for a starting number and an ending number.
2. Ask the user for a number and print its countdown to 1.
3. Ask the user for a number and print all even numbers up to that number.

10. Example 7: Multiple variables in a `for` loop

```
for (int i = 1, j = 10; i <= 5; i++, j--)
{
    cout << i << " " << j << endl;
}
```

Output:

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

11. Example 8: Leave initialization outside the loop

```
int i = 1;

for (; i <= 5; i++)
{
```

```
    cout << i << endl;
}
```

12. Example 9: Leave update statement inside the loop

```
for (int i = 1; i <= 5; )
{
    cout << i << endl;
    i++;
}
```

13. Example 10: Infinite `for` loop

```
for (;;)
{
    cout << "Hello World" << endl;
}
```

This loop runs continuously until it is stopped using `break` or another termination mechanism.

14. Example 11: `for` loop with conditional statement

```
for (int i = 1; i <= 10; i++)
{
    if (i % 2 == 0)
    {
        cout << i << " is Even" << endl;
    }
    else
    {
        cout << i << " is Odd" << endl;
    }
}
```

15. Example 12: Nested `for` Loops – Printing a Rectangle Pattern

A nested loop is a loop placed inside another loop.

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

int main()
{
    for (int i = 1; i <= 4; i++)
    {
        for (int j = 1; j <= 5; j++)
        {
```

```
            cout << "* ";
        }

        cout << endl;
    }

    return 0;
}
```

Output:

```
* * * * *
* * * * *
* * * * *
* * * * *
```

Code Explanation: Read the complete solution with algorithm, pseudocode, dry run, and line-by-line explanation →

Understand the Code

- The outer loop controls the rows. It runs 4 times.
- The inner loop controls the stars in each row. It prints 5 stars.
- After the inner loop finishes, `cout << endl;` starts a new row.
- The inner loop completes all five repetitions for every single outer-loop repetition.

Modify the Code

Try changing the program to produce these patterns.

Pattern A — Rectangle of numbers

```
1 2 3 4
1 2 3 4
1 2 3 4
```

16. Difference Between `while`, `do-while`, and `for`

Feature	`while`	`do-while`	`for`
Condition checked	Before the body	After the body	Before each iteration
Minimum executions	Zero	One	Zero
Initialization and update	Usually separate	Usually separate	Written in the loop header
Common use	Condition-controlled repetition	Menus that run at least once	Counting and sequences

All three loops can solve many of the same problems. Choose the loop that makes the program easiest to understand.

17. Common Mistakes

Mistake 1: Using the wrong condition

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

The condition is false at the beginning, so the body does not execute.

Mistake 2: Forgetting the update

```
for (int i = 1; i <= 5; )
{
    cout << i;
}
```

The counter never changes, so the condition stays true and the loop does not end normally.

Mistake 3: Adding an unnecessary semicolon

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

The semicolon creates an empty loop. The block after it is not the loop body.

18. Practice Coding Exercises

Solve Coding Exercises 20–31 in the `for loop` section. Write, run, and test each C++ program to verify the output.

19. Lesson Summary

- A `for` loop is useful for repeating statements, especially when counting.
- Initialization sets the starting value.
- The condition determines whether the loop continues.
- The update changes the counter.
- Nested loops are useful for patterns, tables, and two-dimensional repetition.