Tishk International University
Faculty of Applied Science
Information Technology Department

Iterative Control Structures (for loop)

Lecture 6

Fall 2025

Course Code: IT117

Grade 1

Islam Abdulazeez
islam.abdulaziz@tiu.edu.iq

January 25, 2026

Programming I

Outlines

- ✓ Increment and decrement operators
- ✓ Compound assignment operators
- ✓ Introduction to loops
- ✓ for loop
- ✓ break and continue statements

Learning Outcomes

- **At the end of today's session, you will be able to:**
  - ✓ Identify the structure of a for loop.
  - ✓ Explain increment, decrement, and compound assignment operators.
  - ✓ Use for loops to control repetition in C++ programs.
  - ✓ Apply break and continue statements appropriately.

The Increment and Decrement Operators

- **Incrementation** and **Decrementation** are operations used to increase or decrease a variable's value.

- These are two ways to increment the variable num

```
num = num + 1;

num += 1;
```

- And num is decremented in both of the following statements:

```
num = num - 1;

num -= 1;
```

The Increment and Decrement Operators

- C++ has operators dedicated to increasing (++) and decreasing (--) variables.

- The following statements use the ++ and -- operator to the variable num:

```
num++;      // Increment by One

num--;     // Decrement by One
```

Post-increment and Pre-increment

Post-increment: (**x++**)

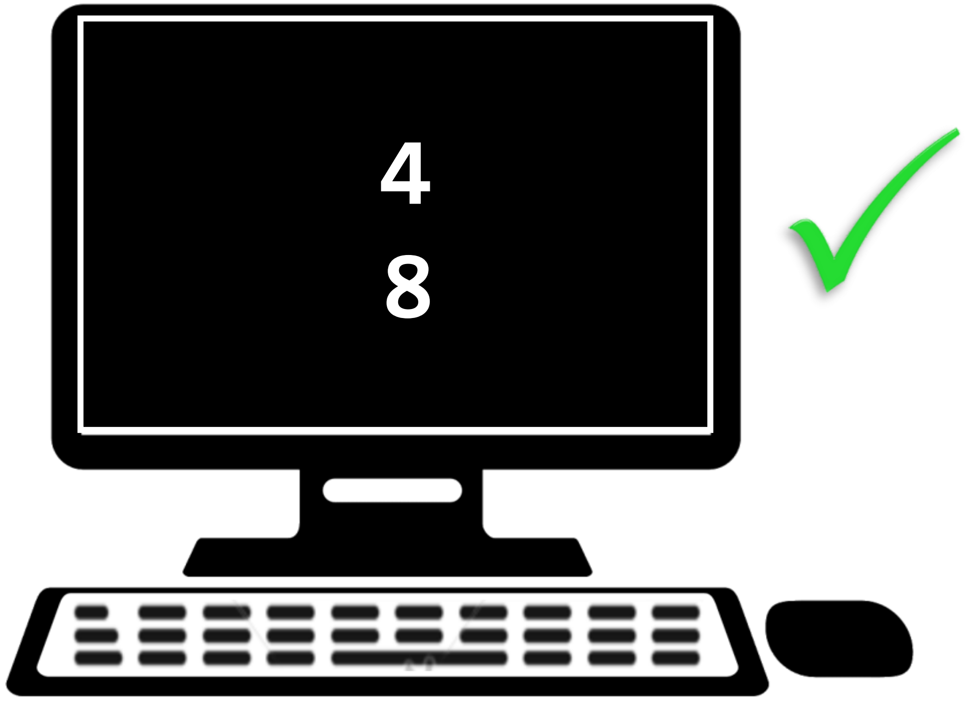
- Operator comes after the operand.
- Operand's value is used first, then it's incremented or decremented.
- The syntax for the **post-increment** and **decrement** operators is x++ and x--, respectively.

Pre-increment: (**++x**)

- Operator comes before the operand.
- Operand is incremented or decremented first, then its updated value is used.
- The syntax for the **pre-increment** and **decrement** operators is ++x and --x, respectively.

Post-increment and Pre-increment (Ex.)

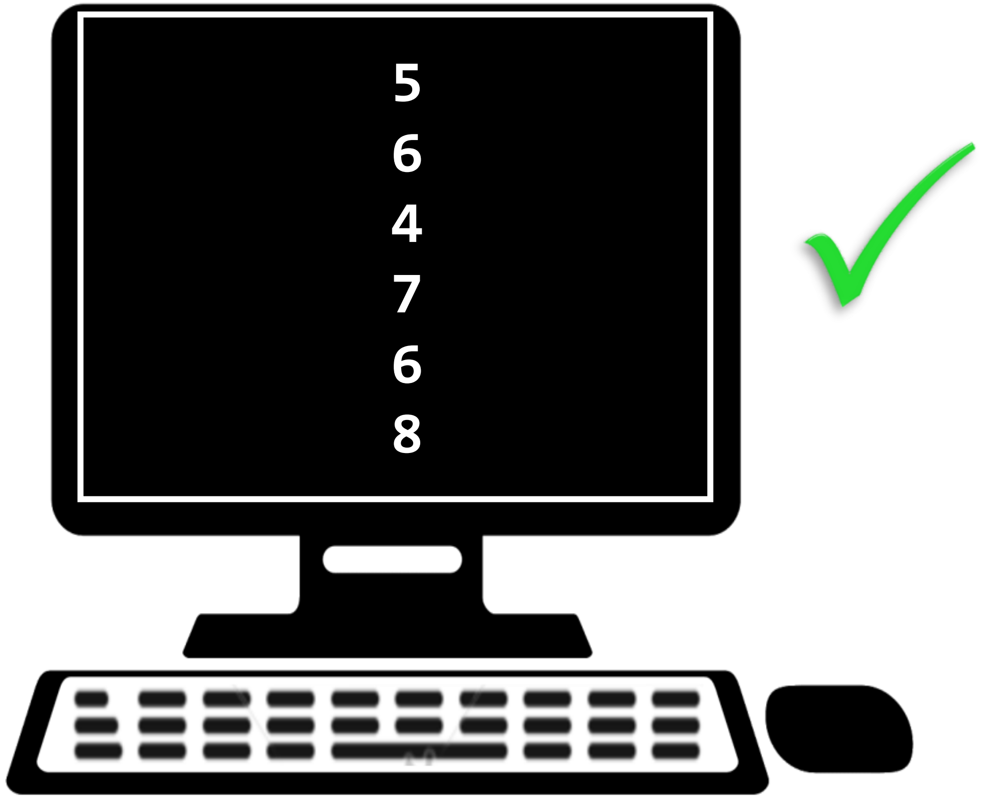
```
int x = 3;
x++;
cout << x << endl;

int y = 9;
y--;
cout << y;
```



```
4
8
```

Post-increment and Pre-increment (Ex.)

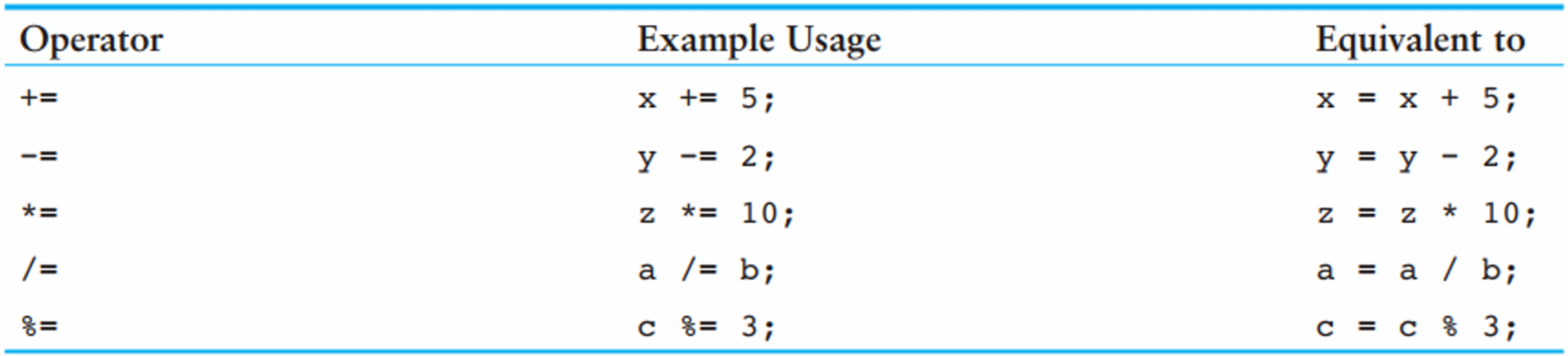
```
int a = 5;
cout << a++ << endl;
cout << a << endl;

int b = 3;
cout << ++b << endl;

int c = 7;
cout << c-- << endl;
cout << c << endl;

int d = 9;
cout << --d;
```



```
5
6
4
7
6
8
```

Combined Assignment

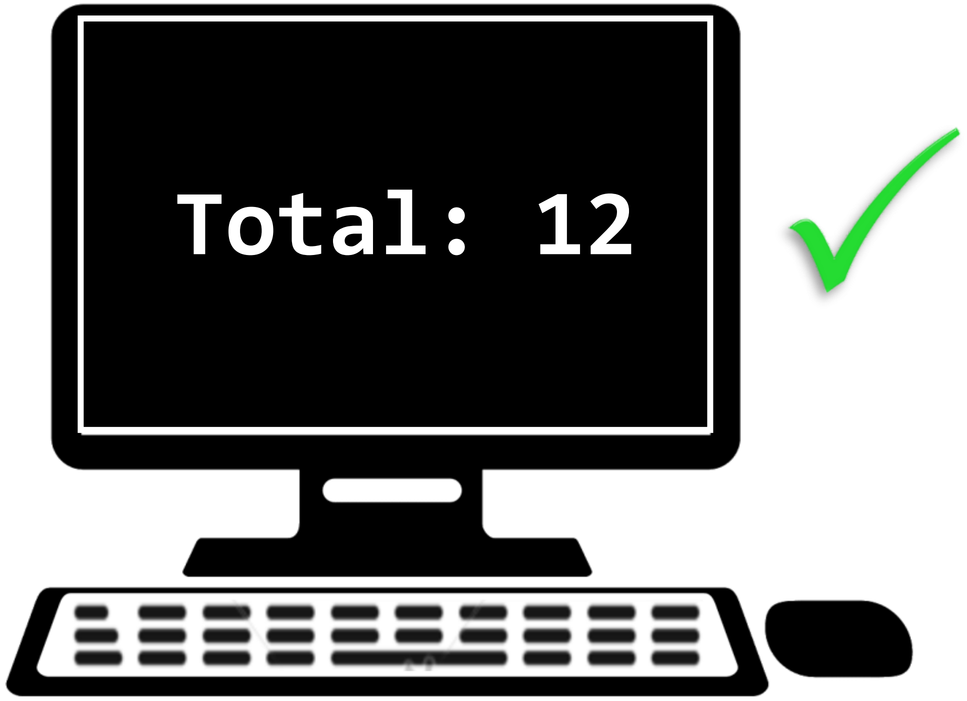
- Combined Assignment, also known as compound assignment, involves combining an arithmetic operation with an assignment.
- It allows you to perform an operation (such as addition, subtraction, multiplication, etc.) and assignment in a single statement



Operator	Example Usage	Equivalent to
`+=`	`x += 5;`	`x = x + 5;`
`-=`	`y -= 2;`	`y = y - 2;`
`*=`	`z *= 10;`	`z = z * 10;`
`/=`	`a /= b;`	`a = a / b;`
`%=`	`c %= 3;`	`c = c % 3;`

Combined Assignment (Ex.)

```
int total = 0;
total += 10;
total += 5;
total -= 3;

cout << "Total: " << total;
```



```
Total: 12
```

Introduction to Loops

- A loop is a control structure that causes a statement or group of statements to repeat.
- C++ has three looping control structures:

✓ for loop ✓ while loop ✓ do-while loop

- The difference between these structures is how they control the repetition.

Why loops?

- **Why Use a Loop?**

  It helps reduce code complexity and improves readability.

```
*
*
*
*
*
```

~~`cout << "*" << endl;`~~
~~`cout << "*" << endl;`~~
~~`cout << "*" << endl;`~~
~~`cout << "*" << endl;`~~
~~`cout << "*" << endl;`~~

```
for (..............................) {
        cout<<"*“<<endl;
}
```

✓

For Loop

- If the number of iteration is fixed, it is recommended to use **for loop**.
- **Syntax**

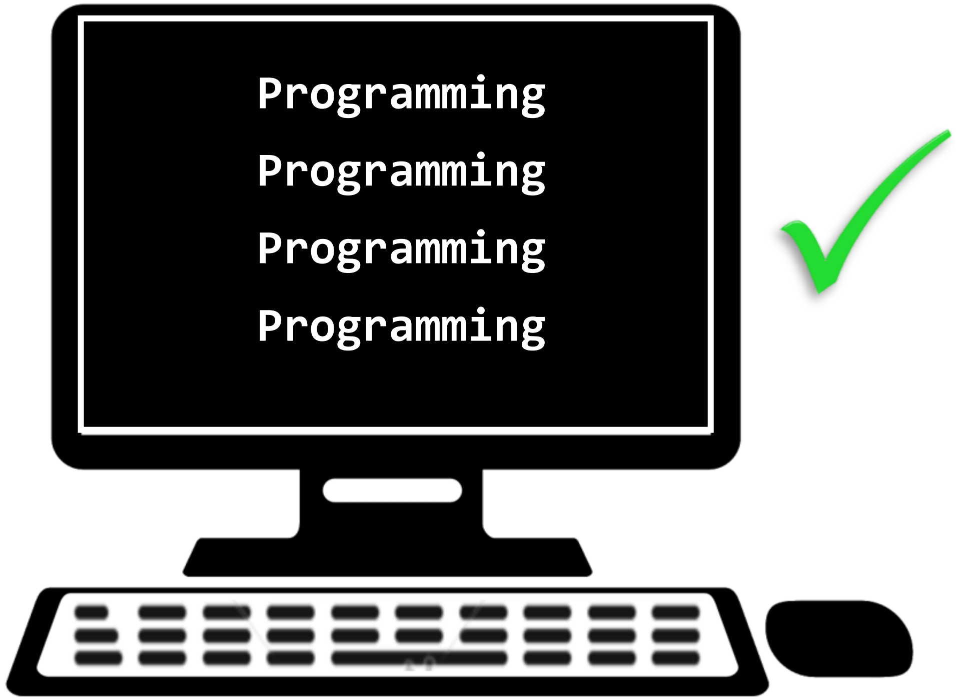
```
for (statement 1; statement 2; statement 3) {
        // code block to be executed
}
```

- **Statement 1:** Setting the variable to the start point.
- **Statement 2:** Defines the condition for executing the code block.
- **Statement 3**: It's the increasing or decreasing step that drives the loop to its end.

For Loop (Ex.)

5

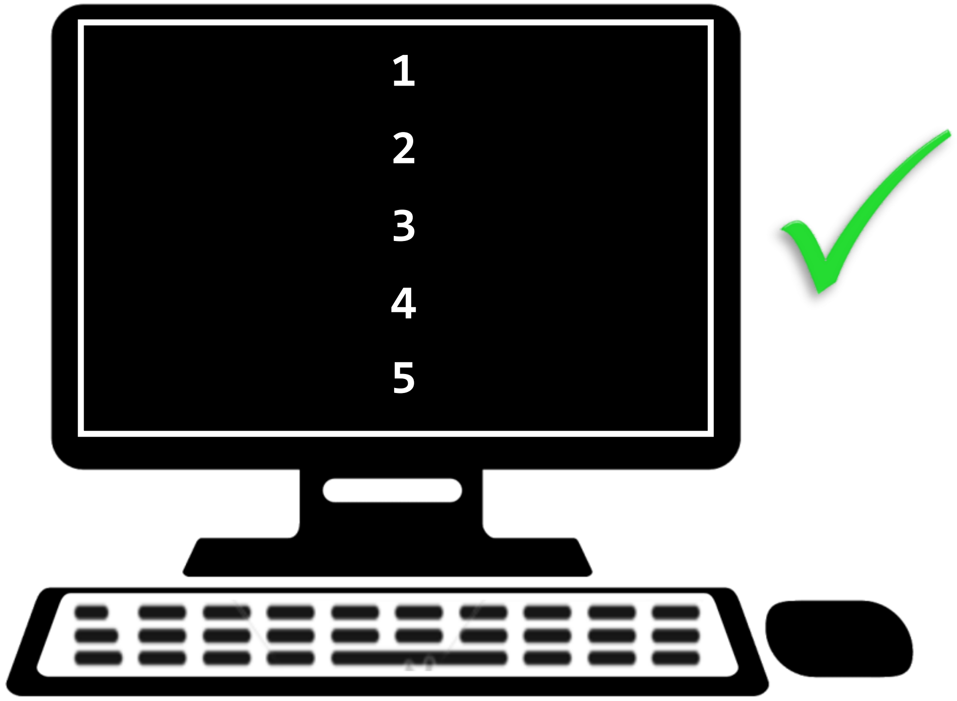
```
for (int i = 1; i < 5; i++) {
    cout<<"Programming"<<endl;
}
```



```
Programming
Programming
Programming
Programming
```

For Loop (Ex.)

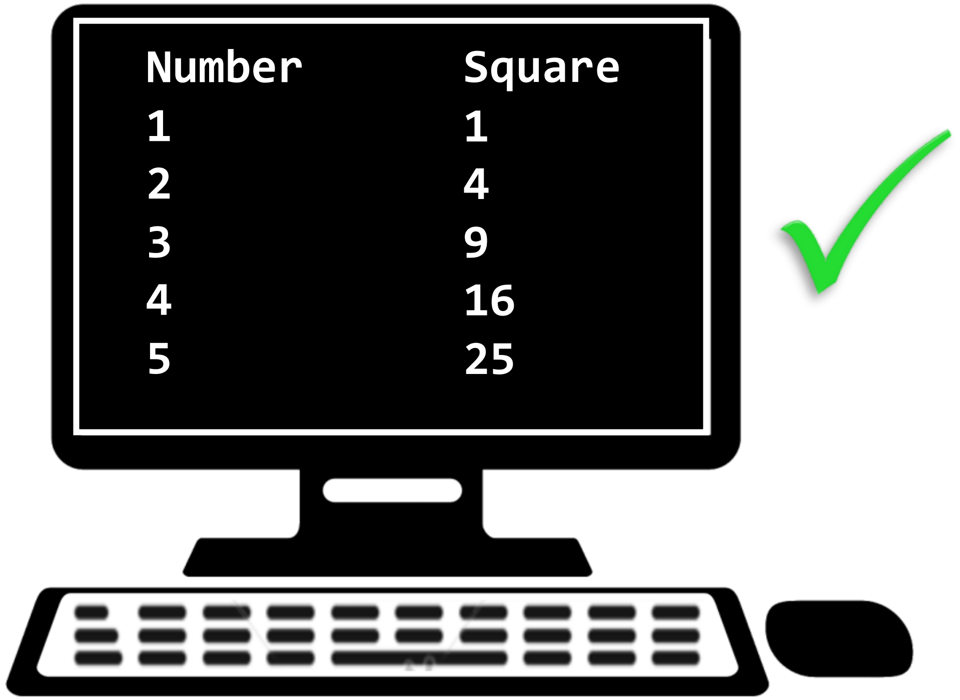
```
for (int i = 1; i <= 5; i++) {
    cout << i << endl;
}
```



```
1
2
3
4
5
```

For Loop (Ex.)

```
cout << "Number\t\tSqaure" << endl;

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



```
Number        Square
1             1
2             4
3             9
4             16
5             25
```

Other Forms of the Update Expression

- Vary the control variable from 1 to 10 in increments of 1.

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

- Vary the control variable from 10 down to 1 in decrements of 1.

```
for (int i = 10; i >= 1; i--)
```

- Vary the control variable over the following sequence of values: 2, 5, 8, 11, 14, 17, 20.

```
for (int i = 2; i <= 20; i+=3)
```

- Vary the control variable from 20 down to 2 in steps of -2.

```
for (int i = 20; i >= 2; i-=2)
```

For Loop (Ex.)

```
for (int i = 1; i <= 10; i += 2) {
    cout << i << " ";
}
```

```
1 3 5 7 9
```

```
for (int i = 10; i >= 1; i -= 2) {
    cout << i << " ";
}
```

```
10 8 6 4 2
```

```
for (int i = 5; i <= 20; i += 5) {
    cout << i << " ";
}
```

```
5 10 15 20
```

For Loop (Ex.)

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

```
1 4 9 16 25
```

```
for (int i = 10; i >= 2; i -= 2) {
    cout << i << " ";
}
```

```
10 8 6 4 2
```

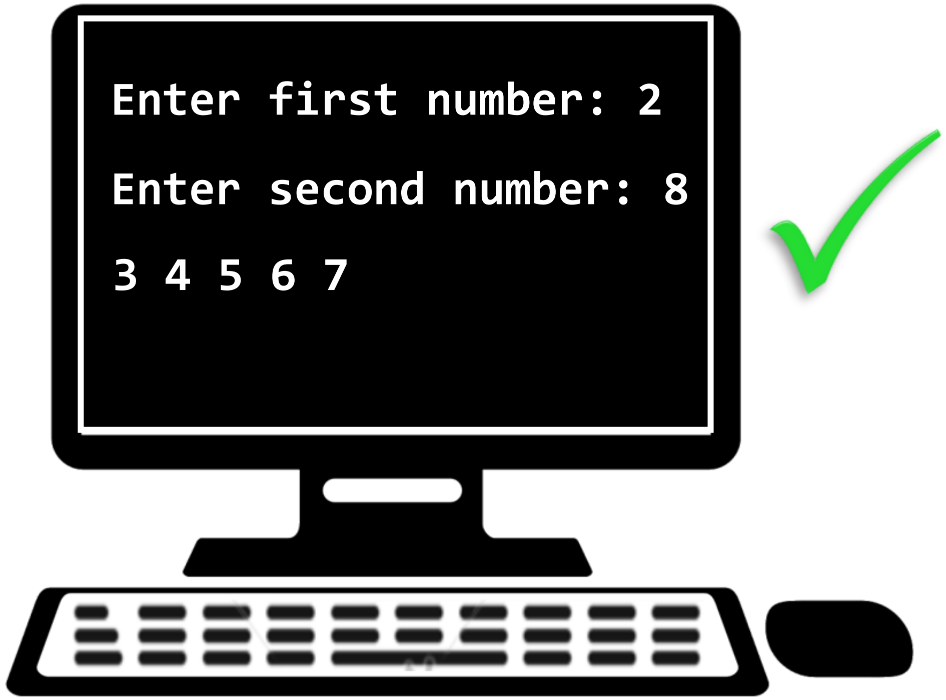
```
for (int i = 3; i <= 15; i += 3) {
    cout << i << " ";
}
```

```
3 6 9 12 15
```

User Controlled for Loop

- Write a C++ program that asks the user to input two numbers and print the numbers between them.

```cpp
int firstNumber, secondNumber;

cout << "Enter first number: ";
cin >> firstNumber;
cout << "Enter second number: ";
cin >> secondNumber;

for (int i = firstNumber+1; i < secondNumber; i++){
    cout << i << " ";
}
```

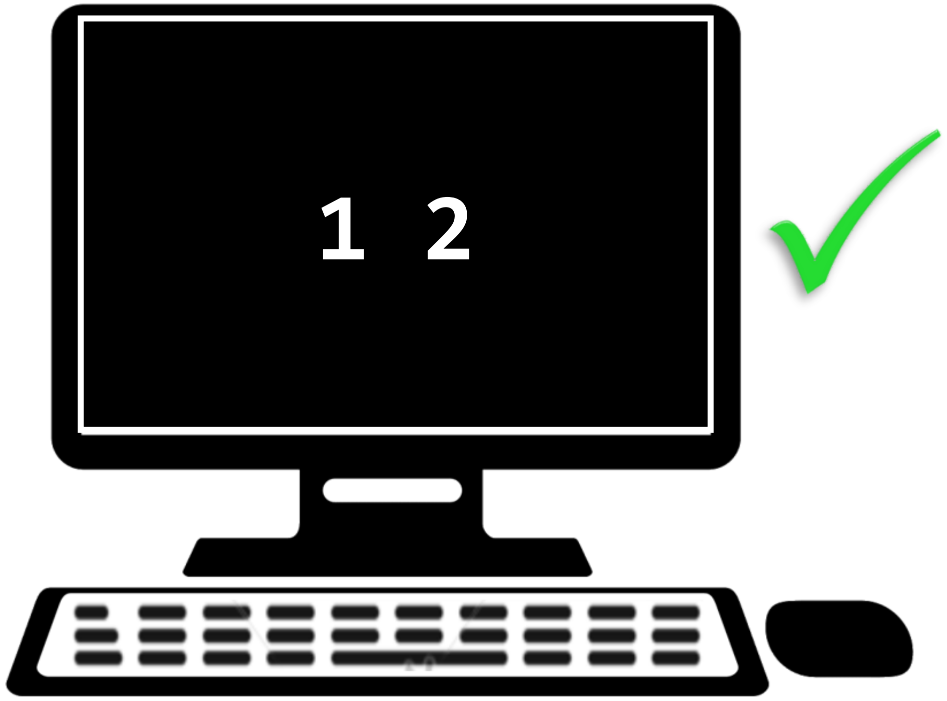


```
Enter first number: 2

Enter second number: 8

3 4 5 6 7
```

Break Statement

- The **break** statement is used to stop a loop immediately.
- When **break** is executed, the program exits the loop and continues with the code after the loop.

```
for (int i = 1; i <= 5; i++) {
     if (i == 3) {
          break;
     }
     cout << i << " ";
}
```



```
1 2
```

Continue Statement

- The **continue** statement is used to skip the current iteration of a loop.
- When **continue** is executed, the loop does not stop; it jumps directly to the next iteration.

```
for (int i = 1; i <= 5; i++) {
    if (i == 3) {
        continue;
    }
    cout << i << " ";
}
```

```
1 2 4 5
```

Activities and Next Lecture's Topic

Activities

- Review this lecture note
- Practice

Next Lecture's Topic

- Iteration Control Structures (while loop and do–while loop)

References

- Gaddis, T. (2014). Starting out with C++: Early objects (7th ed.). Pearson Education.

Thank You!