
🔁 Loops in C++

✅ Purpose of Loops

Loops are used to execute a block of code repeatedly until a certain condition is met. This helps reduce redundancy, increases maintainability, and makes programs efficient.

🔄 Types of Loops in C++

1. **`for` loop**
2. **`while` loop**
3. **`do...while` loop**
4. **Range-based `for` loop (C++11)**
5. **Use of lambdas in loops (C++11/14)**
6. **Structured bindings in loops (C++17)**

🔹 1. `for` Loop (Traditional)

✅ Syntax:

```
for(initialization; condition; increment/decrement) {
    // Loop body
}
```

🔍 Example:

```
#include <iostream>
int main() {
    for(int i = 0; i < 5; ++i) {
        std::cout << "i = " << i << std::endl;
    }
}
```

🔹 2. `while` Loop

✅ Syntax:

```
while(condition) {
    // Loop body
}
```

🔍 Example:

```
int i = 0;
while(i < 5) {
    std::cout << "i = " << i << std::endl;
    ++i;
}
```

🔹 3. `do...while` Loop

✅ **Syntax:**
```
do {
    // Loop body
} while(condition);
```

🔍 **Example:**
```
int i = 0;
do {
    std::cout << "i = " << i << std::endl;
    ++i;
} while(i < 5);
```

🧠 **Note**: `do...while` ensures the body executes **at least once**.

🔹 4. Range-Based `for` Loop (Introduced in C++11)

✅ **Purpose:**

To iterate directly over containers (arrays, vectors, etc.) without needing an index.

✅ **Syntax:**
```
for (declaration : container) {
    // Loop body
}
```

🔍 **Example:**
```
#include <vector>
#include <iostream>

int main() {
    std::vector<int> nums = {10, 20, 30};

    for (int num : nums) {
        std::cout << num << std::endl;
    }
}
```

🧠 **Behind the Scenes:**

It uses `begin()` and `end()` of the container.

🔹 5. auto and const auto& (C++11 & C++14 Enhancement)

🔍 **Example with auto:**

```
for (auto val : nums) {
    std::cout << val << std::endl;
}
```

🔍 **Example with const auto&:**

```
for (const auto& val : nums) {
    std::cout << val << std::endl;
}
```

🧠 **Why use const auto&?** - Prevents unnecessary copying (important for large objects). - Ensures object isn’t accidentally modified.

🔹 6. Using Lambdas in Loops (C++11 & C++14)

Lambdas can be used inside or outside loops for concise logic encapsulation.

🔍 **Example: Lambda in Loop**

```
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    std::vector<int> data = {1, 2, 3, 4, 5};
    std::for_each(data.begin(), data.end(), [](int x) {
        std::cout << "Square: " << x * x << std::endl;
    });
}
```

🔹 7. Structured Bindings (C++17)

Useful when iterating over a map or pair-like container.

☑️ **Syntax:**

```
for (auto& [key, value] : map) {
    // Use key and value directly
}
```

🔍 **Example:**

```
#include <iostream>
#include <map>
```

```cpp
int main() {
    std::map<std::string, int> age = {{"Alice", 30}, {"Bob", 25}};

    for (const auto& [name, years] : age) {
        std::cout << name << " is " << years << " years old.\n";
    }
}
```

🧠 **Benefit**: Cleaner and more expressive than using `first` and `second`.

🔷 Additional Concepts

🔁 Loop Control Statements

- `break`: exits the nearest loop
- `continue`: skips to the next iteration
- `goto`: rarely used; jumps to a labeled statement

🧠 Example with `break` and `continue`:

```cpp
for (int i = 0; i < 10; ++i) {
    if (i == 5) continue;
    if (i == 8) break;
    std::cout << i << " ";
}
// Output: 0 1 2 3 4 6 7
```

🧪 Loop Use Cases and Best Practices

🧰 Use Case 1: Array Processing

```cpp
int arr[] = {1, 2, 3, 4};
for (int val : arr) std::cout << val << " ";
```

🧰 Use Case 2: Searching in Containers

```cpp
bool found = false;
for (int val : nums) {
    if (val == 25) {
        found = true;
        break;
    }
}
```

🧰 Use Case 3: Modifying a Container

```cpp
for (auto& val : nums) {
    val *= 2;
}
```

📌 Summary Table

Feature	Supported In	Benefit
Range-based for loop	C++11	Clean iteration over containers
auto in loops	C++11	Simplifies type declarations
Lambdas in loops	C++11/14	Encapsulate logic inside iteration
Structured bindings	C++17	Clean iteration over maps/pairs

Advanced Tips

✅ Use `reserve()` before loops with vectors to avoid reallocations.

```
std::vector<int> v;
v.reserve(100);
for (int i = 0; i < 100; ++i) v.push_back(i);
```

✅ Prefer `const` auto& when reading values from container to avoid copy overhead.