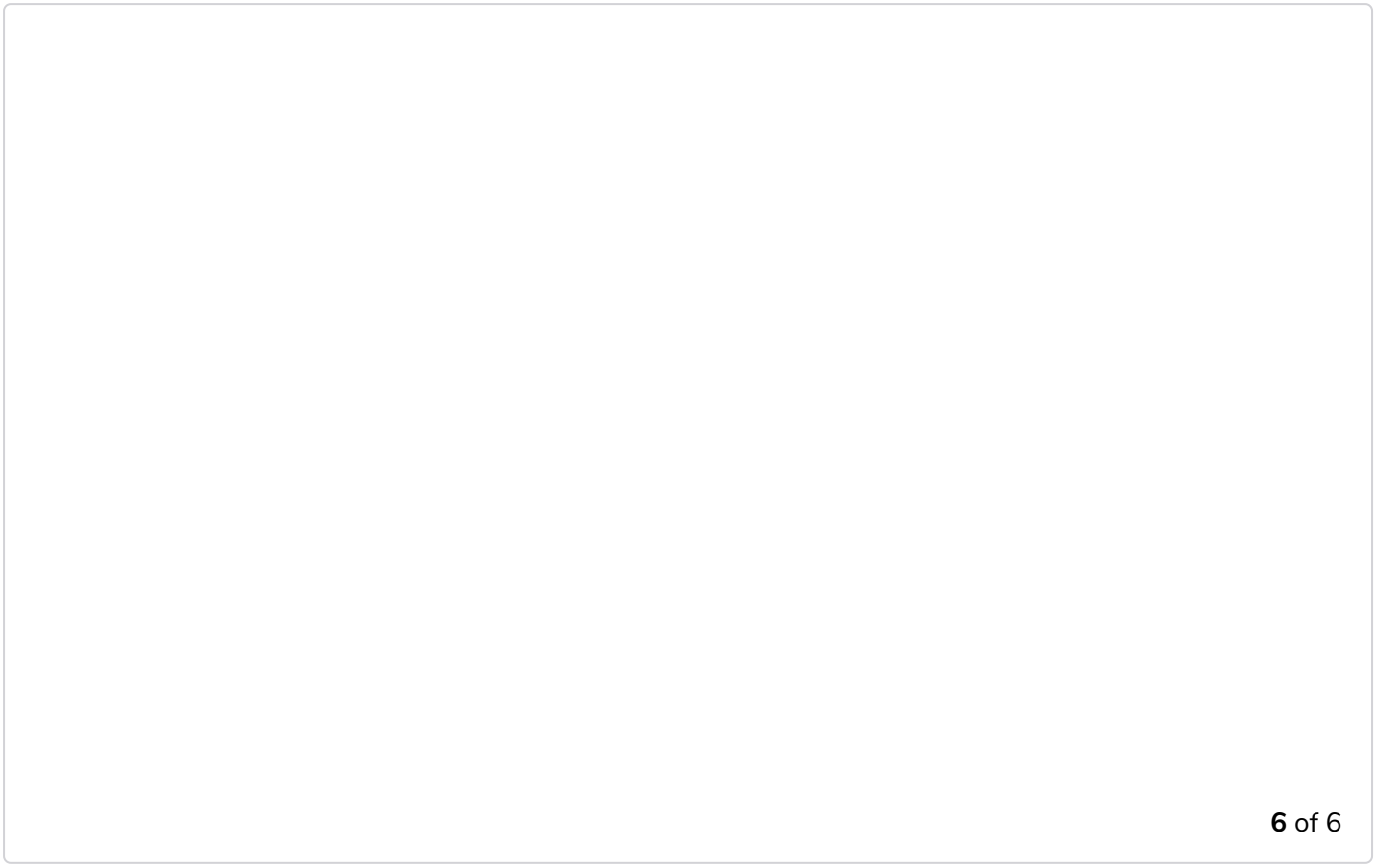
continue Statement

In this lesson, you will be introduced to continue statements in C++.

We'll cover the following

Introduction

Suppose you have a coupon to get five ice-creams free of cost. But the ice-cream man has only three ice-creams. So, when you ask for the fourth one, he tells you he ran out of the ice-cream, and your one coupon is wasted. However, after some time, the ice-creams are restocked, and you can get your free ice-creams.



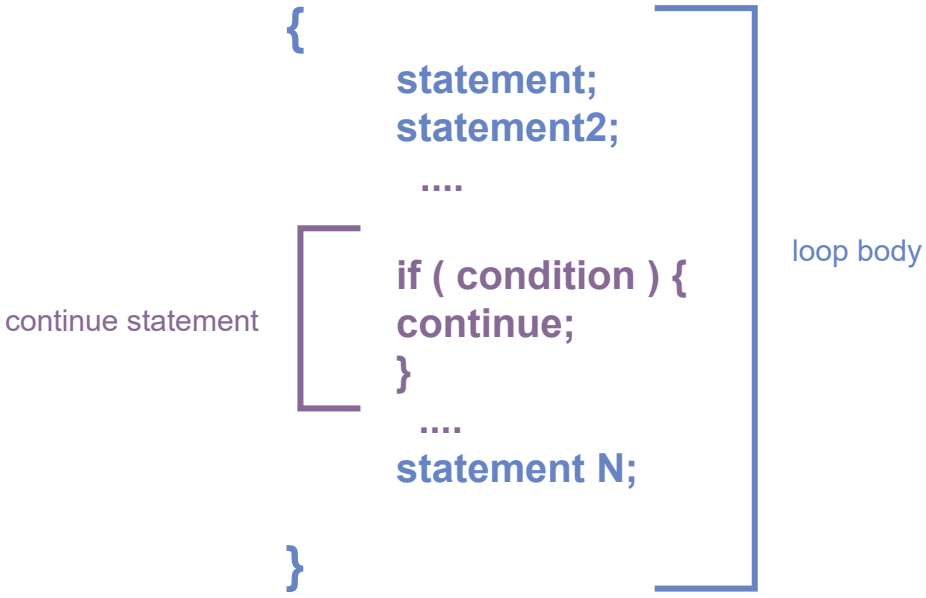
In the era of programming, we can use the `continue` statement for such situations.

---

*The **continue statement** makes the compiler skip the current iteration and move to the next one.*

Use case

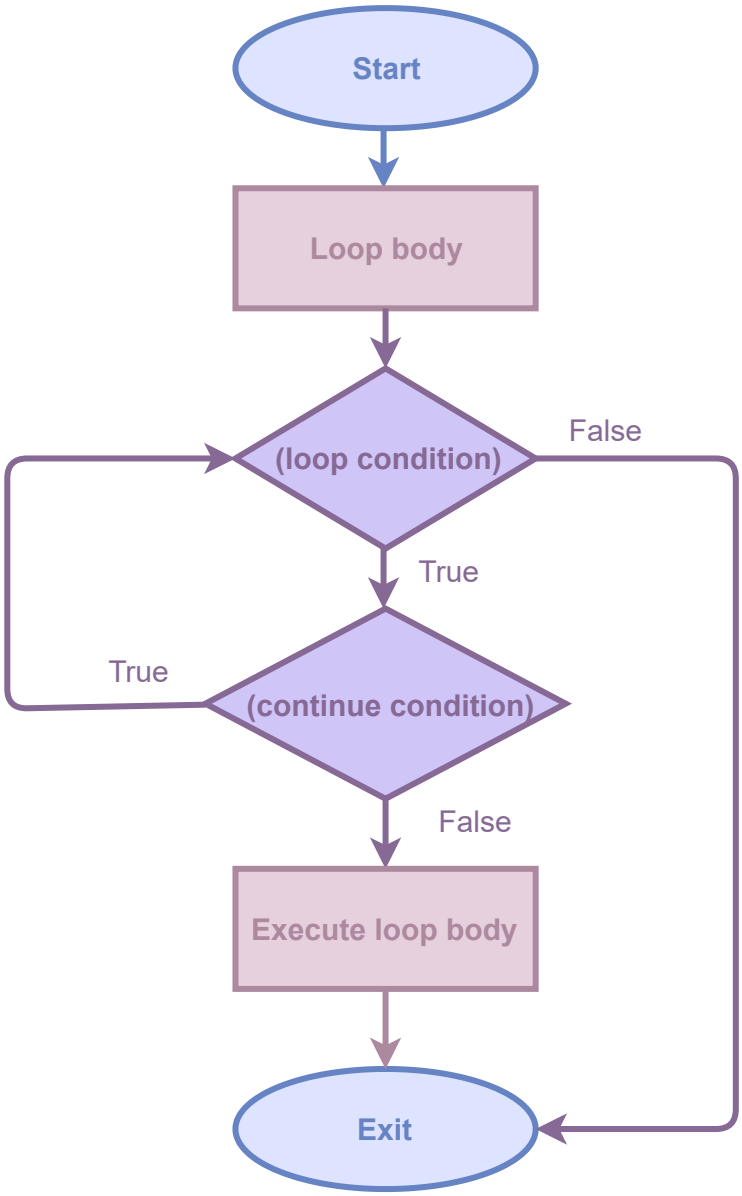
Let's go over the syntax of the `continue` statement. It is so simple to use; just write `continue` before the statements you want to skip in a certain loop iteration!



The basic syntax of a `continue` statement consists of an `if` keyword followed by a condition in round brackets. The curly brackets contain a `continue` keyword that skips the current iteration when the condition evaluates to true.

Flowchart

Let's look at the flowchart of the `continue` statement.



- The loop first evaluates its continuation condition.
- If the condition evaluates to `true`, it executes the code inside the loop. If not, it exits the loop body.
- Inside the loop body, we have the `if` condition followed by a `continue` statement.
- If the `if` condition evaluates to true, it skips the execution of the proceeding statements in the loop body and jumps to the start of the loop for the next iteration. If not, it executes the loop body.

Example program

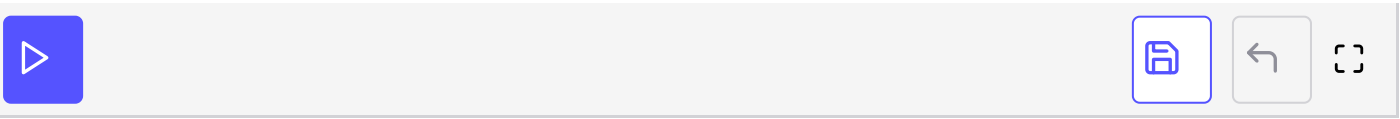
Let's translate the example given above into a C++ program.

Press the **RUN** button and see the output!

```cpp
#include <iostream>

using namespace std;


int main() {
  // Initialize variable icecream
  int icecream;
  // for loop start
  for (icecream = 5; icecream > 0; icecream--) {
    // loop body
    cout << "Number of free ice-creams = " << icecream << endl;
    // continue statement
    if (icecream == 2) {
      cout << "Sorry! We ran out of ice-cream" << endl;
      continue;
    }
    cout << "Buy an icecream" << endl;
  }
  // Exit loop
  return 0;
}
```



Explanation

In the code above, we have a for loop iterating from `5` to `1`. However, since we have a `continue` statement that is executed when the value of the loop variable is `2`, the loop skips this iteration, and it transfers the control to the loop condition.

Line No. 7: Declares a variable `icecream`

Line No. 9:

- `icecream = 5:` The initial value of the `icecream` is set to `5`.
- `icecream > 0:` When the loop condition evaluates to true, it executes the statements from **Line No. 11 to 18**.
- `icecream--:` After executing the loop block, it jumps back to **Line No. 9**, where it decrements the value of the `icecream` by `1` and evaluates the condition again.

Line No. 11: Prints the value of `ice-cream` to the console.

Line No. 13: Checks if the value of the `ice-cream` is `2`. If true, it then executes **Line No. 14** to **Line No. 16**.

Line No. 14: Prints `Sorry! We ran out of ice-cream` to the console

Line No. 15: Exits the loop body and jump to **Line No.9**

Line No. 17: Prints `Buy an icecream` to the console

If `number = 1`, then what is the output of the following code?

```
for (number; number < 4; number += 1) {

  if (number == 2) {
    continue;
  }
  cout << number << endl;

}
```

Retake Quiz

Well, this marks the end of our discussion on loops. Let’s dive right in and solve some challenges related to loops.