# C++ if, if...else and Nested if...else

#### In this tutorial, we will learn about the if...else statement to create decision making programs with the help of examples.

In computer programming, we use the if statement to run a block code only when a certain condition is met.

For example, assigning grades (A, B, C) based on marks obtained by a student.

* if the percentage is above **90**, assign grade **A**
* if the percentage is above **75**, assign grade **B**
* if the percentage is above **65**, assign grade **C**

There are three forms of if...else statements in C++.

1. if statement
2. if...else statement
3. if...else if...else statement

## C++ if Statement

The syntax of the if statement is:

if (condition) {

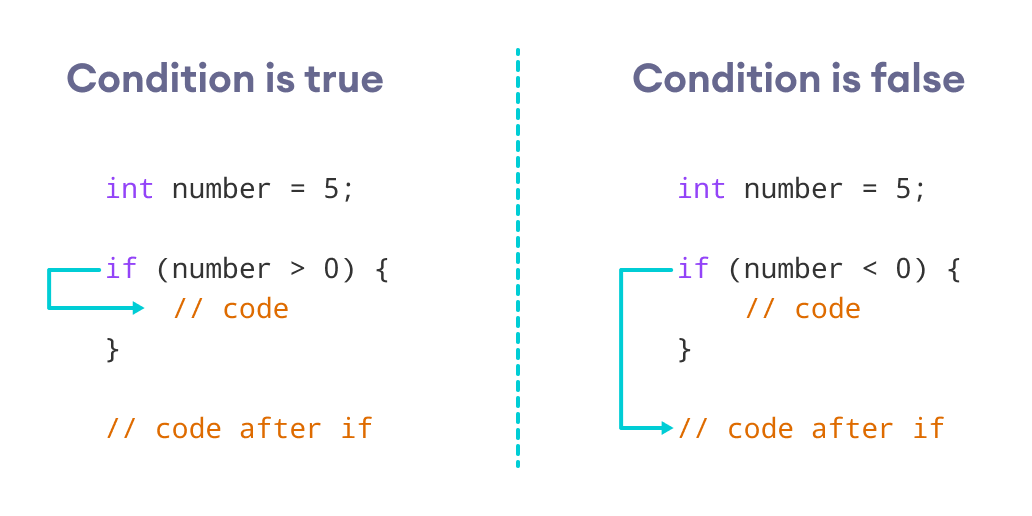
// body of if statement

}

The if statement evaluates the condition inside the parentheses ( ).

* If the condition evaluates to true, the code inside the body of if is executed.
* If the condition evaluates to false, the code inside the body of if is skipped.

**Note:** The code inside { } is the body of the if statement.

Working of C++ if Statement

### Example 1: C++ if Statement

// Program to print positive number entered by the user

// If the user enters a negative number, it is skipped

#include <iostream>

using namespace std;

int main() {

int number;

cout << "Enter an integer: ";

cin >> number;

// checks if the number is positive

if (number > 0) {

cout << "You entered a positive integer: " << number << endl;

}

cout << "This statement is always executed.";

return 0;

}

**Output 1**

Enter an integer: 5

You entered a positive number: 5

This statement is always executed.

When the user enters 5, the condition number > 0 is evaluated to true and the statement inside the body of if is executed.

**Output 2**

Enter a number: -5

This statement is always executed.

When the user enters -5, the condition number > 0 is evaluated to false and the statement inside the body of if is not executed.

## C++ if...else

The if statement can have an optional else clause. Its syntax is:

if (condition) {

// block of code if condition is true

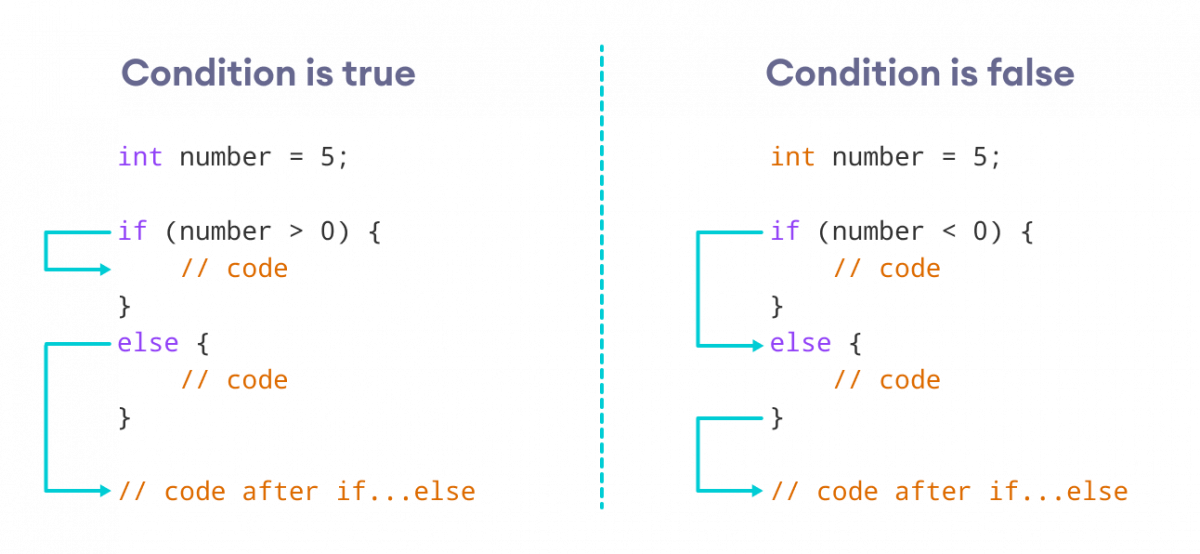
}

else {

// block of code if condition is false

}

The if..else statement evaluates the condition inside the parenthesis.

Working of C++ if...else

If the condition evaluates true,

* the code inside the body of if is executed
* the code inside the body of else is skipped from execution

If the condition evaluates false,

* the code inside the body of else is executed
* the code inside the body of if is skipped from execution

### Example 2: C++ if...else Statement

// Program to check whether an integer is positive or negative

// This program considers 0 as a positive number

#include <iostream>

using namespace std;

int main() {

int number;

cout << "Enter an integer: ";

cin >> number;

if (number >= 0) {

cout << "You entered a positive integer: " << number << endl;

}

else {

cout << "You entered a negative integer: " << number << endl;

}

cout << "This line is always printed.";

return 0;

}

**Output 1**

Enter an integer: 4

You entered a positive integer: 4.

This line is always printed.

In the above program, we have the condition number >= 0. If we enter the number greater or equal to 0, then the condition evaluates true.

Here, we enter 4. So, the condition is true. Hence, the statement inside the body of if is executed.

**Output 2**

Enter an integer: -4

You entered a negative integer: -4.

This line is always printed.

Here, we enter -4. So, the condition is false. Hence, the statement inside the body of else is executed.

## C++ if...else...else if statement

The if...else statement is used to execute a block of code among two alternatives. However, if we need to make a choice between more than two alternatives, we use the if...else if...else statement.

The syntax of the if...else if...else statement is:

if (condition1) {

// code block 1

}

else if (condition2){

// code block 2

}

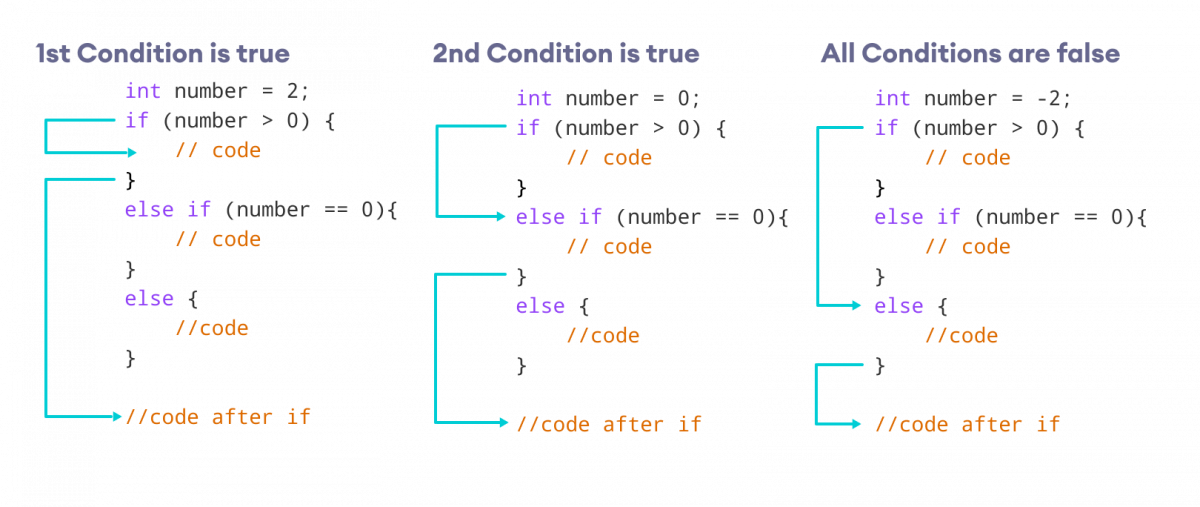
else {

// code block 3

}

Here,

* If condition1 evaluates to true, the code block 1 is executed.
* If condition1 evaluates to false, then condition2 is evaluated.
* If condition2 is true, the code block 2 is executed.
* If condition2 is false, the code block 3 is executed.

How if...else if...else Statement Works

**Note:** There can be more than one else if statement but only one if and else statements.

### Example 3: C++ if...else...else if

// Program to check whether an integer is positive, negative or zero

#include <iostream>

using namespace std;

int main() {

int number;

cout << "Enter an integer: ";

cin >> number;

if (number > 0) {

cout << "You entered a positive integer: " << number << endl;

}

else if (number < 0) {

cout << "You entered a negative integer: " << number << endl;

}

else {

cout << "You entered 0." << endl;

}

cout << "This line is always printed.";

return 0;

}

**Output 1**

Enter an integer: 1

You entered a positive integer: 1.

This line is always printed.

**Output 2**

Enter an integer: -2

You entered a negative integer: -2.

This line is always printed.

**Output 3**

Enter an integer: 0

You entered 0.

This line is always printed.

In this program, we take a number from the user. We then use the if...else if...else ladder to check whether the number is positive, negative, or zero.

If the number is greater than 0, the code inside the if block is executed. If the number is less than 0, the code inside the else if block is executed. Otherwise, the code inside the else block is executed.

## C++ Nested if...else

Sometimes, we need to use an if statement inside another if statement. This is known as nested if statement.

Think of it as multiple layers of if statements. There is a first, outer if statement, and inside it is another, inner if statement. Its syntax is:

// outer if statement

if (condition1) {

// statements

// inner if statement

if (condition2) {

// statements

}

}

**Notes:**

* We can add else and else if statements to the inner if statement as required.
* The inner if statement can also be inserted inside the outer else or else if statements (if they exist).
* We can nest multiple layers of if statements.

### Example 4: C++ Nested if

// C++ program to find if an integer is even or odd or neither (0)

// using nested if statements

#include <iostream>

using namespace std;

int main() {

int num;

cout << "Enter an integer: ";

cin >> num;

// outer if condition

if (num != 0) {

// inner if condition

if ((num % 2) == 0) {

cout << "The number is even." << endl;

}

// inner else condition

else {

cout << "The number is odd." << endl;

}

}

// outer else condition

else {

cout << "The number is 0 and it is neither even nor odd." << endl;

}

cout << "This line is always printed." << endl;

}

**Output 1**

Enter an integer: 34

The number is even.

This line is always printed.

**Output 2**

Enter an integer: 35

The number is odd.

This line is always printed.

**Output 3**

Enter an integer: 0

The number is 0 and it is neither even nor odd.

This line is always printed.

In the above example,

* We take an integer as an input from the user and store it in the variable num.
* We then use an if...else statement to check whether num is not equal to 0.
  + If true, then the **inner** if...else statement is executed.
  + If false, the code inside the **outer** else condition is executed, which prints "The number is 0 and neither even nor odd."
* The **inner** if...else statement checks whether the input number is divisible by 2.
  + If true, then we print a statement saying that the number is even.
  + If false, we print that the number is odd.

Notice that 0 is also divisible by 2, but it is actually not an even number. This is why we first make sure that the input number is not 0 in the outer if condition.

**Note:** As you can see, nested if...else makes your logic complicated. If possible, you should always try to avoid nested if...else.

## Body of if...else With Only One Statement

If the body of if...else has only one statement, you can omit { } in the program. For example, you can replace

int number = 5;

if (number > 0) {

cout << "The number is positive." << endl;

}

else {

cout << "The number is negative." << endl;

}

with

int number = 5;

if (number > 0)

cout << "The number is positive." << endl;

else

cout << "The number is negative." << endl;

The output of both programs will be the same.

**Note:** Although it's not necessary to use { } if the body of if...else has only one statement, using { } makes your code more readable.

## More on Decision Making

In certain situations, a **ternary operator** can replace an if...else statement. To learn more, visit [C++ Ternary Operator](https://www.programiz.com/cpp-programming/cpp-programming/ternary-operator/).

If we need to make a choice between more than one alternatives based on a given test condition, the switch statement can be used. To learn more, visit [C++ switch](https://www.programiz.com/cpp-programming/switch/).

Check out these examples to learn more:

[C++ Program to Check Whether Number is Even or Odd](https://www.programiz.com/cpp-programming/examples/even-odd)

[C++ Program to Check Whether a Character is Vowel or Consonant.](https://www.programiz.com/cpp-programming/examples/vowel-consonant)

[C++ Program to Find Largest Number Among Three Numbers](https://www.programiz.com/cpp-programming/examples/largest-number-among-three)

# C++ for Loop

#### In this tutorial, we will learn about the C++ for loop and its working with the help of some examples.

In computer programming, loops are used to repeat a block of code.

For example, let's say we want to show a message 100 times. Then instead of writing the print statement 100 times, we can use a loop.

That was just a simple example; we can achieve much more efficiency and sophistication in our programs by making effective use of loops.

There are 3 types of loops in C++.

* for loop
* while loop
* do...while loop

This tutorial focuses on C++ for loop. We will learn about the other type of loops in the upcoming tutorials.

## C++ for loop

The syntax of for-loop is:

for (initialization; condition; update) {

// body of-loop

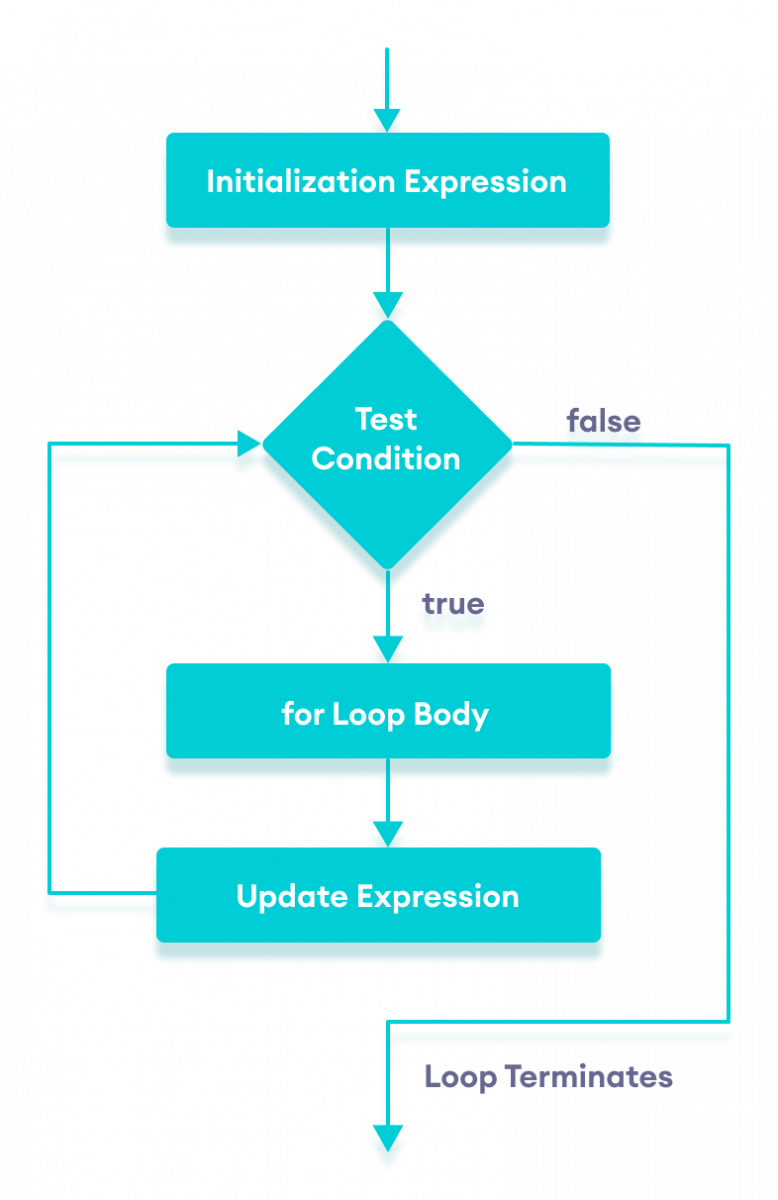
}

Here,

* initialization - initializes variables and is executed only once
* condition - if true, the body of for loop is executed  
  if false, the for loop is terminated
* update - updates the value of initialized variables and again checks the condition

To learn more about conditions, check out our tutorial on [C++ Relational and Logical Operators](https://www.programiz.com/cpp-programming/relational-logical-operators).

## Flowchart of for Loop in C++

Flowchart of for loop in C++

### Example 1: Printing Numbers From 1 to 5

#include <iostream>

using namespace std;

int main() {

for (int i = 1; i <= 5; ++i) {

cout << i << " ";

}

return 0;

}

**Output**

1 2 3 4 5

Here is how this program works

|  |  |  |  |
| --- | --- | --- | --- |
| Iteration | Variable | i <= 5 | Action |
| 1st | i = 1 | true | 1 is printed. i is increased to 2. |
| 2nd | i = 2 | true | 2 is printed. i is increased to 3. |
| 3rd | i = 3 | true | 3 is printed. i is increased to 4. |
| 4th | i = 4 | true | 4 is printed. i is increased to 5. |
| 5th | i = 5 | true | 5 is printed. i is increased to 6. |
| 6th | i = 6 | false | The loop is terminated |

### Example 2: Display a text 5 times

// C++ Program to display a text 5 times

#include <iostream>

using namespace std;

int main() {

for (int i = 1; i <= 5; ++i) {

cout << "Hello World! " << endl;

}

return 0;

}

**Output**

Hello World!

Hello World!

Hello World!

Hello World!

Hello World!

Here is how this program works

|  |  |  |  |
| --- | --- | --- | --- |
| Iteration | Variable | i <= 5 | Action |
| 1st | i = 1 | true | Hello World! is printed and i is increased to 2. |
| 2nd | i = 2 | true | Hello World! is printed and i is increased to 3. |
| 3rd | i = 3 | true | Hello World! is printed and i is increased to 4. |
| 4th | i = 4 | true | Hello World! is printed and i is increased to 5. |
| 5th | i = 5 | true | Hello World! is printed and i is increased to 6. |
| 6th | i = 6 | false | The loop is terminated |

### Example 3: Find the sum of first n Natural Numbers

// C++ program to find the sum of first n natural numbers

// positive integers such as 1,2,3,...n are known as natural numbers

#include <iostream>

using namespace std;

int main() {

int num, sum;

sum = 0;

cout << "Enter a positive integer: ";

cin >> num;

for (int count = 1; count <= num; ++count) {

sum += count;

}

cout << "Sum = " << sum << endl;

return 0;

}

**Output**

Enter a positive integer: 10

Sum = 55

In the above example, we have two variables num and sum. The sum variable is assigned with 0 and the num variable is assigned with the value provided by the user.

Note that we have used a for loop.

for(int count = 1; count <= num; ++count)

Here,

* int count = 1: initializes the count variable
* count <= num: runs the loop as long as count is less than or equal to num
* ++count: increase the count variable by 1 in each iteration

When count becomes 11, the condition is false and sum will be equal to 0 + 1 + 2 + ... + 10.

## Ranged Based for Loop

In C++11, a new range-based for loop was introduced to work with collections such as **arrays** and **vectors**. Its syntax is:

for (variable : collection) {

// body of loop

}

Here, for every value in the collection, the for loop is executed and the value is assigned to the variable.

### Example 4: Range Based for Loop

#include <iostream>

using namespace std;

int main() {

int num\_array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

for (int n : num\_array) {

cout << n << " ";

}

return 0;

}

**Output**

1 2 3 4 5 6 7 8 9 10

In the above program, we have declared and initialized an int array named num\_array. It has 10 items.

Here, we have used a range-based for loop to access all the items in the array.

### C++ Infinite for loop

If the condition in a for loop is always true, it runs forever (until memory is full). For example,

// infinite for loop

for(int i = 1; i > 0; i++) {

// block of code

}

In the above program, the condition is always true which will then run the code for infinite times.

Check out these examples to learn more:

* [C++ Program to Calculate Sum of Natural Numbers](https://www.programiz.com/cpp-programming/examples/sum-natural-number)
* [C++ Program to Find Factorial](https://www.programiz.com/cpp-programming/examples/factorial)
* [C++ Program to Generate Multiplication Table](https://www.programiz.com/cpp-programming/examples/multiplication-table)

In the next tutorial, we will learn about while and do...while loop.

# C++ while and do...while Loop

#### In this tutorial, we will learn the use of while and do...while loops in C++ programming with the help of some examples.

In computer programming, loops are used to repeat a block of code.

For example, let's say we want to show a message 100 times. Then instead of writing the print statement 100 times, we can use a loop.

That was just a simple example; we can achieve much more efficiency and sophistication in our programs by making effective use of loops.

There are **3** types of loops in C++.

1. for loop
2. while loop
3. do...while loop

In the previous tutorial, we learned about the [C++ for loop](https://www.programiz.com/cpp-programming/for-loop). Here, we are going to learn about while and do...while loops.

## C++ while Loop

The syntax of the while loop is:

while (condition) {

// body of the loop

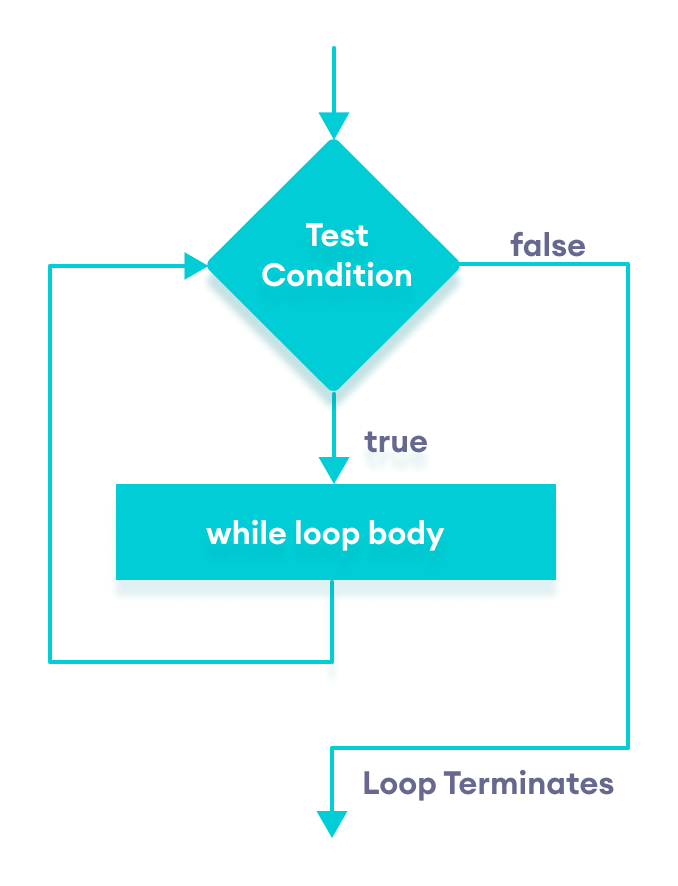
}

Here,

* A while loop evaluates the condition
* If the condition evaluates to true, the code inside the while loop is executed.
* The condition is evaluated again.
* This process continues until the condition is false.
* When the condition evaluates to false, the loop terminates.

To learn more about the conditions, visit [C++ Relational and Logical Operators](https://www.programiz.com/cpp-programming/relational-logical-operators).

### Flowchart of while Loop

Flowchart of C++ while loop

### Example 1: Display Numbers from 1 to 5

// C++ Program to print numbers from 1 to 5

#include <iostream>

using namespace std;

int main() {

int i = 1;

// while loop from 1 to 5

while (i <= 5) {

cout << i << " ";

++i;

}

return 0;

}

**Output**

1 2 3 4 5

Here is how the program works.

|  |  |  |  |
| --- | --- | --- | --- |
| Iteration | Variable | i <= 5 | Action |
| 1st | i = 1 | true | 1 is printed and i is increased to 2. |
| 2nd | i = 2 | true | 2 is printed and i is increased to 3. |
| 3rd | i = 3 | true | 3 is printed and i is increased to 4 |
| 4th | i = 4 | true | 4 is printed and i is increased to 5. |
| 5th | i = 5 | true | 5 is printed and i is increased to 6. |
| 6th | i = 6 | false | The loop is terminated |

### Example 2: Sum of Positive Numbers Only

// program to find the sum of positive numbers

// if the user enters a negative number, the loop ends

// the negative number entered is not added to the sum

#include <iostream>

using namespace std;

int main() {

int number;

int sum = 0;

// take input from the user

cout << "Enter a number: ";

cin >> number;

while (number >= 0) {

// add all positive numbers

sum += number;

// take input again if the number is positive

cout << "Enter a number: ";

cin >> number;

}

// display the sum

cout << "\nThe sum is " << sum << endl;

return 0;

}

**Output**

Enter a number: 6

Enter a number: 12

Enter a number: 7

Enter a number: 0

Enter a number: -2

The sum is 25

In this program, the user is prompted to enter a number, which is stored in the variable number.

In order to store the sum of the numbers, we declare a variable sum and initialize it to the value of 0.

The while loop continues until the user enters a negative number. During each iteration, the number entered by the user is added to the sum variable.

When the user enters a negative number, the loop terminates. Finally, the total sum is displayed.

## C++ do...while Loop

The do...while loop is a variant of the while loop with one important difference: the body of do...while loop is executed once before the condition is checked.

Its syntax is:

do {

// body of loop;

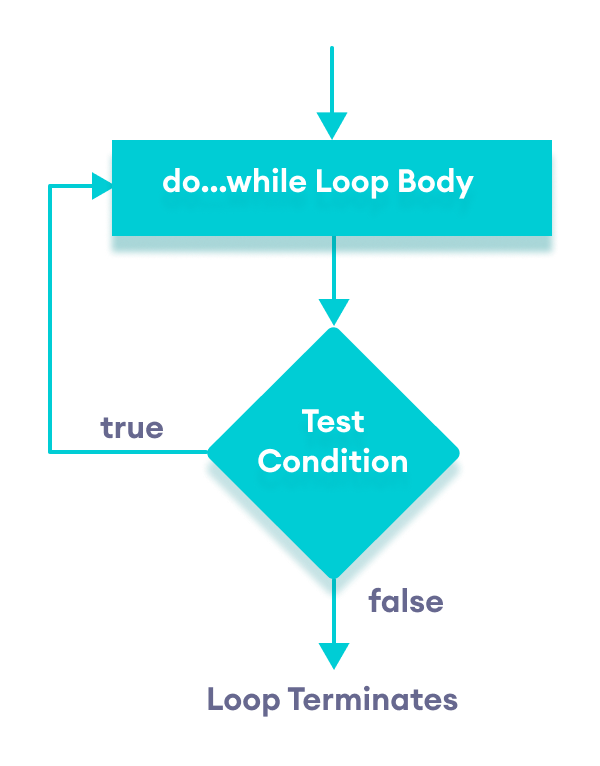
}

while (condition);

Here,

* The body of the loop is executed at first. Then the condition is evaluated.
* If the condition evaluates to true, the body of the loop inside the do statement is executed again.
* The condition is evaluated once again.
* If the condition evaluates to true, the body of the loop inside the do statement is executed again.
* This process continues until the condition evaluates to false. Then the loop stops.

### Flowchart of do...while Loop

Flowchart of C++ do...while loop

### Example 3: Display Numbers from 1 to 5

// C++ Program to print numbers from 1 to 5

#include <iostream>

using namespace std;

int main() {

int i = 1;

// do...while loop from 1 to 5

do {

cout << i << " ";

++i;

}

while (i <= 5);

return 0;

}

**Output**

1 2 3 4 5

Here is how the program works.

|  |  |  |  |
| --- | --- | --- | --- |
| Iteration | Variable | i <= 5 | Action |
|  | i = 1 | not checked | 1 is printed and i is increased to 2 |
| 1st | i = 2 | true | 2 is printed and i is increased to 3 |
| 2nd | i = 3 | true | 3 is printed and i is increased to 4 |
| 3rd | i = 4 | true | 4 is printed and i is increased to 5 |
| 4th | i = 5 | true | 5 is printed and i is increased to **6** |
| 5th | i = 6 | false | The loop is terminated |

### Example 4: Sum of Positive Numbers Only

// program to find the sum of positive numbers

// If the user enters a negative number, the loop ends

// the negative number entered is not added to the sum

#include <iostream>

using namespace std;

int main() {

int number = 0;

int sum = 0;

do {

sum += number;

// take input from the user

cout << "Enter a number: ";

cin >> number;

}

while (number >= 0);

// display the sum

cout << "\nThe sum is " << sum << endl;

return 0;

}

**Output 1**

Enter a number: 6

Enter a number: 12

Enter a number: 7

Enter a number: 0

Enter a number: -2

The sum is 25

Here, the do...while loop continues until the user enters a negative number. When the number is negative, the loop terminates; the negative number is not added to the sum variable.

**Output 2**

Enter a number: -6

The sum is 0.

The body of the do...while loop runs only once if the user enters a negative number.

## Infinite while loop

If the condition of a loop is always true, the loop runs for infinite times (until the memory is full). For example,

// infinite while loop

while(true) {

// body of the loop

}

Here is an example of an infinite do...while loop.

// infinite do...while loop

int count = 1;

do {

// body of loop

}

while(count == 1);

In the above programs, the condition is always true. Hence, the loop body will run for infinite times.

## for vs while loops

A for loop is usually used when the number of iterations is known. For example,

// This loop is iterated 5 times

for (int i = 1; i <=5; ++i) {

// body of the loop

}

Here, we know that the for-loop will be executed 5 times.

However, while and do...while loops are usually used when the number of iterations is unknown. For example,

while (condition) {

// body of the loop

}

Check out these examples to learn more:

* [C++ Program to Display Fibonacci Series](https://www.programiz.com/cpp-programming/examples/fibonacci-series)
* [C++ Program to Find GCD](https://www.geeksforgeeks.org/c-program-find-gcd-hcf-two-numbers/)
* [C++ Program to Find LCM](https://www.programiz.com/cpp-programming/examples/lcm)

# C++ break Statement

#### In this tutorial, we will learn about the break statement and its working in loops with the help of examples.

In C++, the break statement terminates the loop when it is encountered.

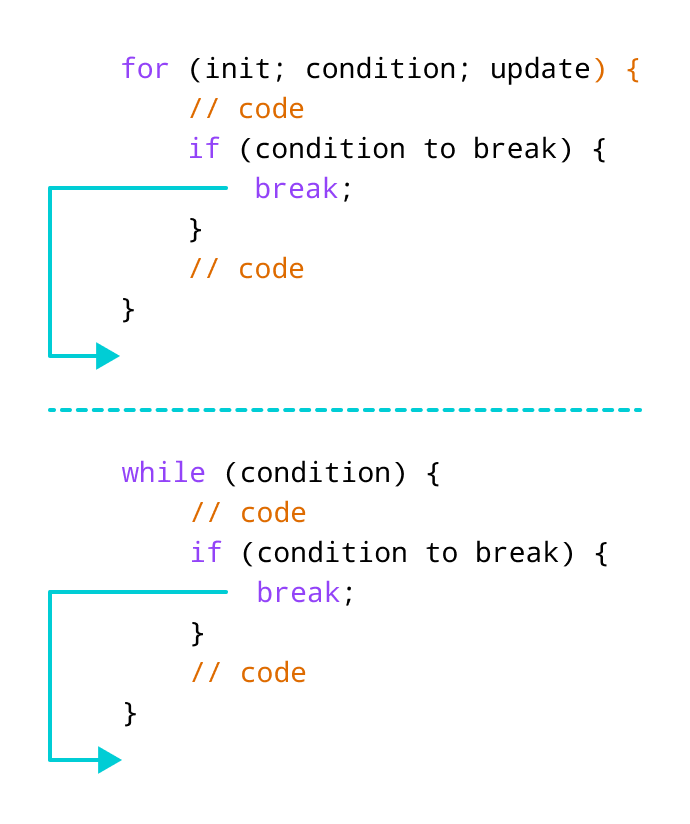
The syntax of the break statement is:

break;

Before you learn about the break statement, make sure you know about:

* [C++ for loop](https://www.programiz.com/cpp-programming/for-loop)
* [C++ if...else](https://www.programiz.com/cpp-programming/for-loop)
* [C++ while loop](https://www.programiz.com/cpp-programming/do-while-loop)

## Working of C++ break Statement

Working of break statement in C++

## Example 1: break with for loop

// program to print the value of i

#include <iostream>

using namespace std;

int main() {

for (int i = 1; i <= 5; i++) {

// break condition

if (i == 3) {

break;

}

cout << i << endl;

}

return 0;

}

**Output**

1

2

In the above program, the for loop is used to print the value of i in each iteration. Here, notice the code:

if (i == 3) {

break;

}

This means, when i is equal to **3**, the break statement terminates the loop. Hence, the output doesn't include values greater than or equal to 3.

Note: The break statement is usually used with decision-making statements.

## Example 2: break with while loop

// program to find the sum of positive numbers

// if the user enters a negative numbers, break ends the loop

// the negative number entered is not added to sum

#include <iostream>

using namespace std;

int main() {

int number;

int sum = 0;

while (true) {

// take input from the user

cout << "Enter a number: ";

cin >> number;

// break condition

if (number < 0) {

break;

}

// add all positive numbers

sum += number;

}

// display the sum

cout << "The sum is " << sum << endl;

return 0;

}

**Output**

Enter a number: 1

Enter a number: 2

Enter a number: 3

Enter a number: -5

The sum is 6.

In the above program, the user enters a number. The while loop is used to print the total sum of numbers entered by the user. Here, notice the code,

if(number < 0) {

break;

}

This means, when the user enters a negative number, the break statement terminates the loop and codes outside the loop are executed.

The while loop continues until the user enters a negative number.

## break with Nested loop

When break is used with nested loops, break terminates the inner loop. For example,

// using break statement inside

// nested for loop

#include <iostream>

using namespace std;

int main() {

int number;

int sum = 0;

// nested for loops

// first loop

for (int i = 1; i <= 3; i++) {

// second loop

for (int j = 1; j <= 3; j++) {

if (i == 2) {

break;

}

cout << "i = " << i << ", j = " << j << endl;

}

}

return 0;

}

**Output**

i = 1, j = 1

i = 1, j = 2

i = 1, j = 3

i = 3, j = 1

i = 3, j = 2

i = 3, j = 3

In the above program, the break statement is executed when i == 2. It terminates the inner loop, and the control flow of the program moves to the outer loop.

Hence, the value of i = 2 is never displayed in the output.

The break statement is also used with the switch statement. To learn more, visit [C++ switch statement](https://www.programiz.com/cpp-programming/switch-case).

# C++ continue Statement

#### In this tutorial, we will learn about the continue statement and its working with loops with the help of examples.

In computer programming, the continue statement is used to skip the current iteration of the loop and the control of the program goes to the next iteration.

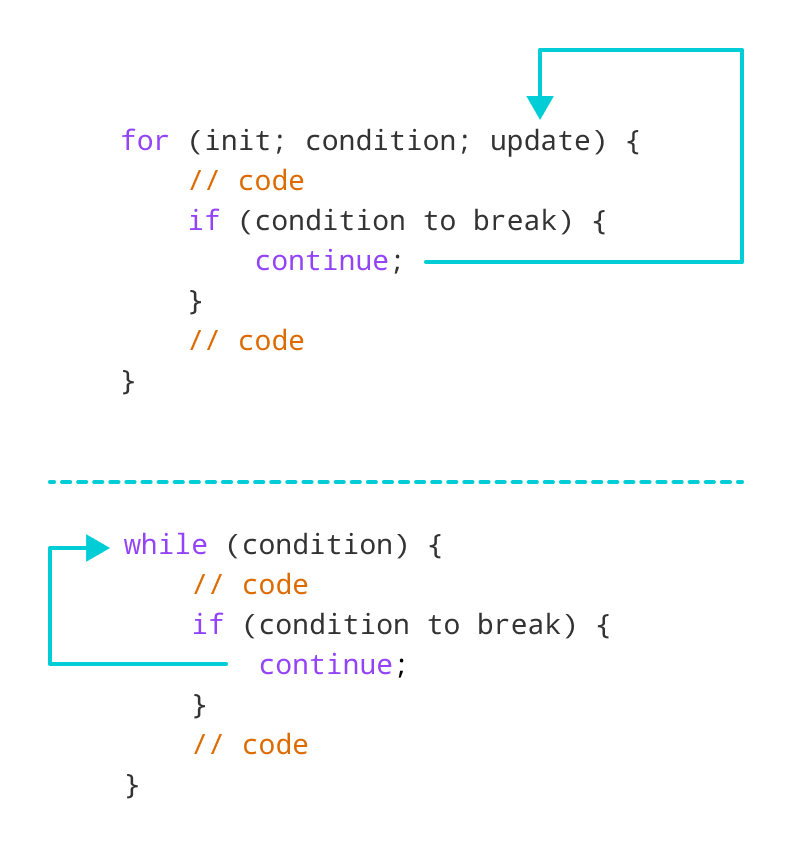
The syntax of the continue statement is:

continue;

Before you learn about the continue statement, make sure you know about,

* [C++ for loop](https://www.programiz.com/cpp-programming/for-loop)
* [C++ if...else](https://www.programiz.com/cpp-programming/for-loop)
* [C++ while loop](https://www.programiz.com/cpp-programming/do-while-loop)

## Working of C++ continue Statement

Working of continue statement in C++

## Example 1: continue with for loop

In a for loop, continue skips the current iteration and the control flow jumps to the update expression.

// program to print the value of i

#include <iostream>

using namespace std;

int main() {

for (int i = 1; i <= 5; i++) {

// condition to continue

if (i == 3) {

continue;

}

cout << i << endl;

}

return 0;

}

**Output**

1

2

4

5

In the above program, we have used the the for loop to print the value of i in each iteration. Here, notice the code,

if (i == 3) {

continue;

}

This means

* When i is equal to 3, the continue statement skips the current iteration and starts the next iteration
* Then, i becomes 4**,** and the condition is evaluated again.
* Hence, 4 and 5 are printed in the next two iterations.

**Note**: The continue statement is almost always used with decision-making statements.

**Note**: The break statement terminates the loop entirely. However, the continue statement only skips the current iteration.

## Example 2: continue with while loop

In a while loop, continue skips the current iteration and control flow of the program jumps back to the while condition.

// program to calculate positive numbers till 50 only

// if the user enters a negative number,

// that number is skipped from the calculation

// negative number -> loop terminate

// numbers above 50 -> skip iteration

#include <iostream>

using namespace std;

int main() {

int sum = 0;

int number = 0;

while (number >= 0) {

// add all positive numbers

sum += number;

// take input from the user

cout << "Enter a number: ";

cin >> number;

// continue condition

if (number > 50) {

cout << "The number is greater than 50 and won't be calculated." << endl;

number = 0; // the value of number is made 0 again

continue;

}

}

// display the sum

cout << "The sum is " << sum << endl;

return 0;

}

**Output**

Enter a number: 12

Enter a number: 0

Enter a number: 2

Enter a number: 30

Enter a number: 50

Enter a number: 56

The number is greater than 50 and won't be calculated.

Enter a number: 5

Enter a number: -3

The sum is 99

In the above program, the user enters a number. The while loop is used to print the total sum of positive numbers entered by the user, as long as the numbers entered are not greater than 50.

Notice the use of the continue statement.

if (number > 50){

continue;

}

* When the user enters a number greater than 50, the continue statement skips the current iteration. Then the control flow of the program goes to the condition of while loop.
* When the user enters a number less than 0, the loop terminates.

**Note**: The continue statement works in the same way for the do...while loops.

## continue with Nested loop

When continue is used with nested loops, it skips the current iteration of the inner loop. For example,

// using continue statement inside

// nested for loop

#include <iostream>

using namespace std;

int main() {

int number;

int sum = 0;

// nested for loops

// first loop

for (int i = 1; i <= 3; i++) {

// second loop

for (int j = 1; j <= 3; j++) {

if (j == 2) {

continue;

}

cout << "i = " << i << ", j = " << j << endl;

}

}

return 0;

}

**Output**

i = 1, j = 1

i = 1, j = 3

i = 2, j = 1

i = 2, j = 3

i = 3, j = 1

i = 3, j = 3

In the above program, when the continue statement executes, it skips the current iteration in the inner loop. And the control of the program moves to the **update expression** of the inner loop.

Hence, the value of j = 2 is never displayed in the output.

**C++ switch..case Statement**

In this tutorial, we will learn about switch statement and its working in C++ programming with the help of some examples.

The switch statement allows us to execute a block of code among many alternatives.

The syntax of the switch statement in C++ is:

switch (expression) {

case constant1:

// code to be executed if

// expression is equal to constant1;

break;

case constant2:

// code to be executed if

// expression is equal to constant2;

break;

.

.

.

default:

// code to be executed if

// expression doesn't match any constant

}

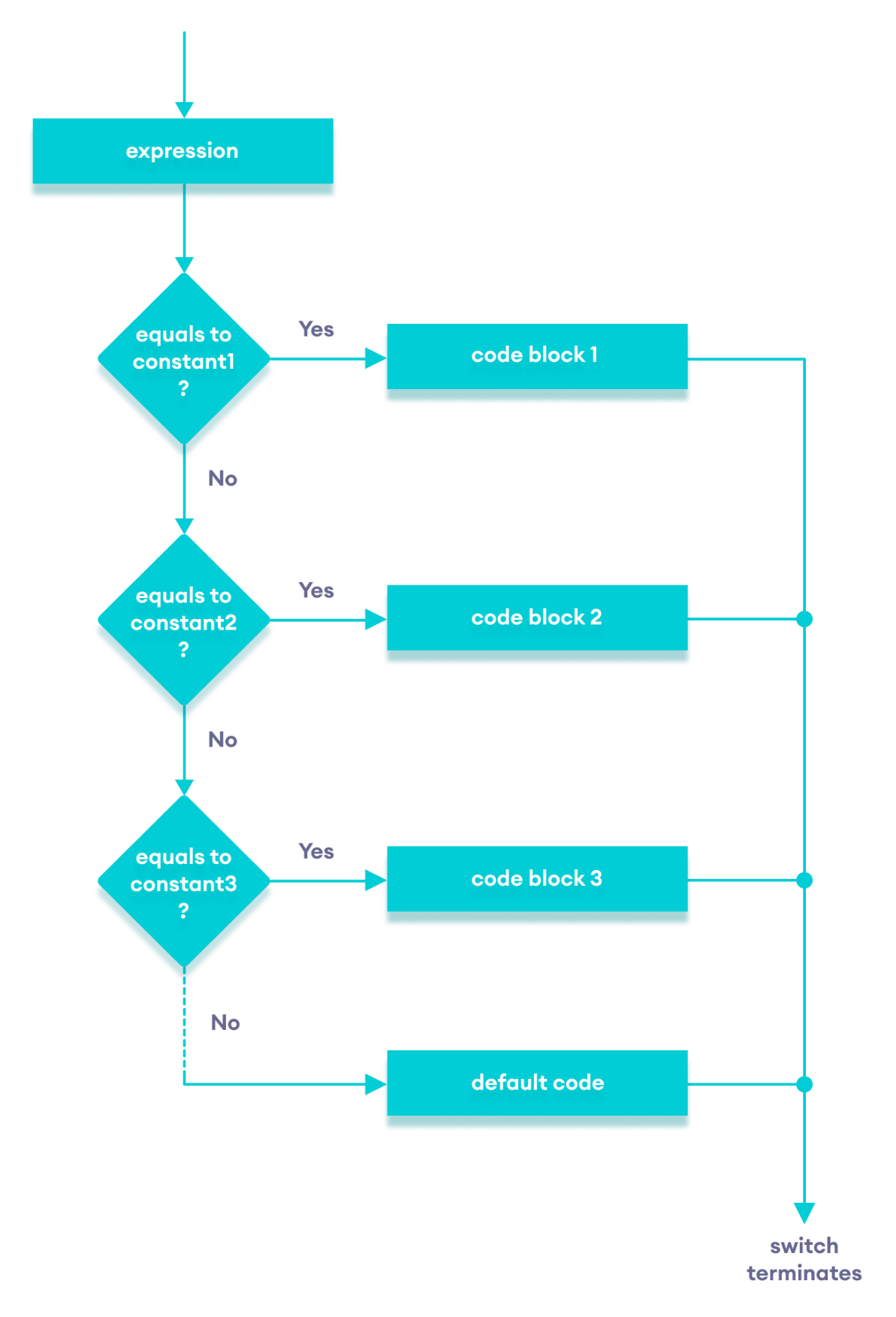
**How does the switch statement work?**

The expression is evaluated once and compared with the values of each case label.

* If there is a match, the corresponding code after the matching label is executed. For example, if the value of the variable is equal to constant2, the code after case constant2: is executed until the break statement is encountered.
* If there is no match, the code after default: is executed.

**Note**: We can do the same thing with the if...else..if ladder. However, the syntax of the switch statement is cleaner and much easier to read and write.

**Flowchart of switch Statement**

Flowchart of C++ switch...case statement

**Example: Create a Calculator using the switch Statement**

// Program to build a simple calculator using switch Statement

#include <iostream>

using namespace std;

int main() {

char oper;

float num1, num2;

cout << "Enter an operator (+, -, \*, /): ";

cin >> oper;

cout << "Enter two numbers: " << endl;

cin >> num1 >> num2;

switch (oper) {

case '+':

cout << num1 << " + " << num2 << " = " << num1 + num2;

break;

case '-':

cout << num1 << " - " << num2 << " = " << num1 - num2;

break;

case '\*':

cout << num1 << " \* " << num2 << " = " << num1 \* num2;

break;

case '/':

cout << num1 << " / " << num2 << " = " << num1 / num2;

break;

default:

// operator is doesn't match any case constant (+, -, \*, /)

cout << "Error! The operator is not correct";

break;

}

return 0;

}

**Output 1**

Enter an operator (+, -, \*, /): +

Enter two numbers:

2.3

4.5

2.3 + 4.5 = 6.8

**Output 2**

Enter an operator (+, -, \*, /): -

Enter two numbers:

2.3

4.5

2.3 - 4.5 = -2.2

**Output 3**

Enter an operator (+, -, \*, /): \*

Enter two numbers:

2.3

4.5

2.3 \* 4.5 = 10.35

**Output 4**

Enter an operator (+, -, \*, /): /

Enter two numbers:

2.3

4.5

2.3 / 4.5 = 0.511111

**Output 5**

Enter an operator (+, -, \*, /): ?

Enter two numbers:

2.3

4.5

Error! The operator is not correct.

In the above program, we are using the switch...case statement to perform addition, subtraction, multiplication, and division.

**How This Program Works**

1. We first prompt the user to enter the desired operator. This input is then stored in the char variable named oper.
2. We then prompt the user to enter two numbers, which are stored in the float variables num1 and num2.
3. The switch statement is then used to check the operator entered by the user:
   * If the user enters +, addition is performed on the numbers.
   * If the user enters -, subtraction is performed on the numbers.
   * If the user enters \*, multiplication is performed on the numbers.
   * If the user enters /, division is performed on the numbers.
   * If the user enters any other character, the default code is printed.

Notice that the break statement is used inside each case block. This terminates the switch statement.

If the break statement is not used, all cases after the correct case are executed.

# C++ goto Statement

#### In this article, you'll learn about goto statment, how it works and why should it be avoided.

In C++ programming, goto statement is used for altering the normal sequence of program execution by transferring control to some other part of the program.

## Syntax of goto Statement

goto label;

... .. ...

... .. ...

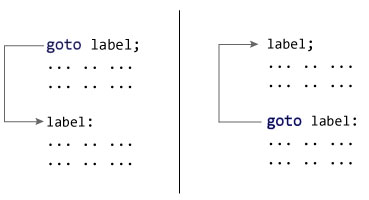
... .. ...

label:

statement;

... .. ...

In the syntax above, label is an identifier. When goto label; is encountered, the control of program jumps to label: and executes the code below it.



### Example: goto Statement

// This program calculates the average of numbers entered by user.

// If user enters negative number, it ignores the number and

// calculates the average of number entered before it.

# include <iostream>

using namespace std;

int main()

{

float num, average, sum = 0.0;

int i, n;

cout << "Maximum number of inputs: ";

cin >> n;

for(i = 1; i <= n; ++i)

{

cout << "Enter n" << i << ": ";

cin >> num;

if(num < 0.0)

{

// Control of the program move to jump:

goto jump;

}

sum += num;

}

jump:

average = sum / (i - 1);

cout << "\nAverage = " << average;

return 0;

}

**Output**

Maximum number of inputs: 10

Enter n1: 2.3

Enter n2: 5.6

Enter n3: -5.6

Average = 3.95

You can write any C++ program without the use of goto statement and is generally considered a good idea not to use them.

### Reason to Avoid goto Statement

The goto statement gives power to jump to any part of program but, makes the logic of the program complex and tangled.

In modern programming, goto statement is considered a harmful construct and a bad programming practice.

The goto statement can be replaced in most of C++ program with the use of [break and continue statements](https://www.programiz.com/cpp-programming/break-continue).

**[Previ](https://www.programiz.com/cpp-programming/switch-case" \o "C++ switch Statement)**