**C if...else Statement**

In this tutorial, you will learn about the if statement (including if...else and nested if..else) in C programming with the help of examples.

## C if Statement

The syntax of the if statement in C programming is:

if (test expression)

{

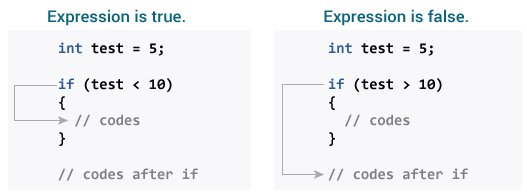
// code

}

### How if statement works?

The if statement evaluates the test expression inside the parenthesis ().

* If the test expression is evaluated to true, statements inside the body of if are executed.
* If the test expression is evaluated to false, statements inside the body of if are not executed.

Working of if Statement

To learn more about when test expression is evaluated to true (non-zero value) and false (0), check [relational](https://www.programiz.com/c-programming/c-operators#relational) and [logical operators](https://www.programiz.com/c-programming/c-operators#logical).

### Example 1: if statement

// Program to display a number if it is negative

#include <stdio.h>

int main() {

int number;

printf("Enter an integer: ");

scanf("%d", &number);

// true if number is less than 0

if (number < 0) {

printf("You entered %d.\n", number);

}

printf("The if statement is easy.");

return 0;

}

**Output 1**

Enter an integer: -2

You entered -2.

The if statement is easy.

When the user enters -2, the test expression number<0 is evaluated to true. Hence, You entered -2 is displayed on the screen.

**Output 2**

Enter an integer: 5

The if statement is easy.

When the user enters 5, the test expression number<0 is evaluated to false and the statement inside the body of if is not executed

## C if...else Statement

The if statement may have an optional else block. The syntax of the if..else statement is:

if (test expression) {

// run code if test expression is true

}

else {

// run code if test expression is false

}

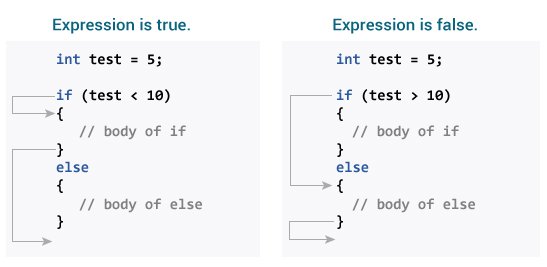
### How if...else statement works?

If the test expression is evaluated to true,

* statements inside the body of if are executed.
* statements inside the body of else are skipped from execution.

If the test expression is evaluated to false,

* statements inside the body of else are executed
* statements inside the body of if are skipped from execution.

Working of if...else Statement

### Example 2: if...else statement

// Check whether an integer is odd or even

#include <stdio.h>

int main() {

int number;

printf("Enter an integer: ");

scanf("%d", &number);

// True if the remainder is 0

if (number%2 == 0) {

printf("%d is an even integer.",number);

}

else {

printf("%d is an odd integer.",number);

}

return 0;

}

**Output**

Enter an integer: 7

7 is an odd integer.

When the user enters 7, the test expression number%2==0 is evaluated to false. Hence, the statement inside the body of else is executed.

## C if...else Ladder

The if...else statement executes two different codes depending upon whether the test expression is true or false. Sometimes, a choice has to be made from more than 2 possibilities.

The if...else ladder allows you to check between multiple test expressions and execute different statements.

### Syntax of if...else Ladder

if (test expression1) {

// statement(s)

}

else if(test expression2) {

// statement(s)

}

else if (test expression3) {

// statement(s)

}

.

.

else {

// statement(s)

}

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

// Program to relate two integers using =, > or < symbol

#include <stdio.h>

int main() {

int number1, number2;

printf("Enter two integers: ");

scanf("%d %d", &number1, &number2);

//checks if the two integers are equal.

if(number1 == number2) {

printf("Result: %d = %d",number1,number2);

}

//checks if number1 is greater than number2.

else if (number1 > number2) {

printf("Result: %d > %d", number1, number2);

}

//checks if both test expressions are false

else {

printf("Result: %d < %d",number1, number2);

}

return 0;

}

**Output**

Enter two integers: 12

23

Result: 12 < 23

## Nested if...else

It is possible to include an if...else statement inside the body of another if...else statement.

### Example 4: Nested if...else

This program given below relates two integers using either <, > and = similar to the if...else ladder's example. However, we will use a nested if...else statement to solve this problem.

#include <stdio.h>

int main() {

int number1, number2;

printf("Enter two integers: ");

scanf("%d %d", &number1, &number2);

if (number1 >= number2) {

if (number1 == number2) {

printf("Result: %d = %d",number1,number2);

}

else {

printf("Result: %d > %d", number1, number2);

}

}

else {

printf("Result: %d < %d",number1, number2);

}

return 0;

}

If the body of an if...else statement has only one statement, you do not need to use brackets {}.

For example, this code

if (a > b) {

printf("Hello");

}

printf("Hi");

is equivalent to

if (a > b)

printf("Hello");

printf("Hi");

# C for Loop

In this tutorial, you will learn to create for loop in C programming with the help of examples.

In programming, a loop is used to repeat a block of code until the specified condition is met.

C programming has three types of loops:

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

We will learn about for loop in this tutorial. In the next tutorial, we will learn about while and do...while loop.

## for Loop

The syntax of the for loop is:

for (initializationStatement; testExpression; updateStatement)

{

// statements inside the body of loop

}

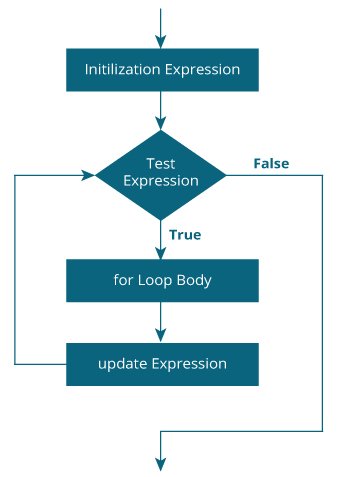
### How for loop works?

* The initialization statement is executed only once.
* Then, the test expression is evaluated. If the test expression is evaluated to false, the for loop is terminated.
* However, if the test expression is evaluated to true, statements inside the body of the for loop are executed, and the update expression is updated.
* Again the test expression is evaluated.

This process goes on until the test expression is false. When the test expression is false, the loop terminates.

To learn more about test expression (when the test expression is evaluated to true and false), check out [relational](https://www.programiz.com/c-programming/c-operators#relational) and [logical operators](https://www.programiz.com/c-programming/c-operators#logical).

### for loop Flowchart

Working of for loop

### Example 1: for loop

// Print numbers from 1 to 10

#include <stdio.h>

int main() {

int i;

for (i = 1; i < 11; ++i)

{

printf("%d ", i);

}

return 0;

}

**Output**

1 2 3 4 5 6 7 8 9 10

1. i is initialized to 1.
2. The test expression i < 11 is evaluated. Since 1 less than 11 is true, the body of for loop is executed. This will print the **1** (value of i) on the screen.
3. The update statement ++i is executed. Now, the value of i will be 2. Again, the test expression is evaluated to true, and the body of for loop is executed. This will print **2** (value of i) on the screen.
4. Again, the update statement ++i is executed and the test expression i < 11 is evaluated. This process goes on until i becomes 11.
5. When i becomes 11, i < 11 will be false, and the for loop terminates.

### Example 2: for loop

// Program to calculate the sum of first n natural numbers

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

#include <stdio.h>

int main()

{

int num, count, sum = 0;

printf("Enter a positive integer: ");

scanf("%d", &num);

// for loop terminates when num is less than count

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

{

sum += count;

}

printf("Sum = %d", sum);

return 0;

}

**Output**

Enter a positive integer: 10

Sum = 55

The value entered by the user is stored in the variable num. Suppose, the user entered 10.

The count is initialized to 1 and the test expression is evaluated. Since the test expression count<=num (1 less than or equal to 10) is true, the body of for loop is executed and the value of sum will equal to 1.

Then, the update statement ++count is executed and count will equal to 2. Again, the test expression is evaluated. Since 2 is also less than 10, the test expression is evaluated to true and the body of the for loop is executed. Now, sum will equal 3.

This process goes on and the sum is calculated until the count reaches 11.

When the count is 11, the test expression is evaluated to 0 (false), and the loop terminates.

Then, the value of sum is printed on the screen.

# C while and do...while Loop

In this tutorial, you will learn to create while and do...while loop in C programming with the help of examples.

In programming, loops are used to repeat a block of code until a specified condition is met.

C programming has three types of loops.

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

In the previous tutorial, we learned about for loop. In this tutorial, we will learn about while and do..while loop.

## while loop

The syntax of the while loop is:

while (testExpression) {

// the body of the loop

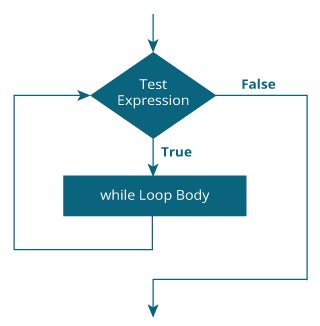
}

### How while loop works?

* The while loop evaluates the testExpression inside the parentheses ().
* If testExpression is **true**, statements inside the body of while loop are executed. Then, testExpression is evaluated again.
* The process goes on until testExpression is evaluated to **false**.
* If testExpression is **false**, the loop terminates (ends).

To learn more about test expressions (when testExpression is evaluated to **true** and **false**), check out [relational](https://www.programiz.com/c-programming/c-operators#relational) and [logical operators](https://www.programiz.com/c-programming/c-operators#logical).

### Flowchart of while loop

Working of while loop

### Example 1: while loop

// Print numbers from 1 to 5

#include <stdio.h>

int main() {

int i = 1;

while (i <= 5) {

printf("%d\n", i);

++i;

}

return 0;

}

**Output**

1

2

3

4

5

Here, we have initialized i to 1.

1. When i = 1, the test expression i <= 5 is **true**. Hence, the body of the while loop is executed. This prints 1 on the screen and the value of i is increased to 2.
2. Now, i = 2, the test expression i <= 5 is again **true**. The body of the while loop is executed again. This prints 2 on the screen and the value of i is increased to 3.
3. This process goes on until i becomes 6. Then, the test expression i <= 5 will be **false** and the loop terminates.

## do...while loop

The do..while loop is similar to the while loop with one important difference. The body of do...while loop is executed at least once. Only then, the test expression is evaluated.

The syntax of the do...while loop is:

do {

// the body of the loop

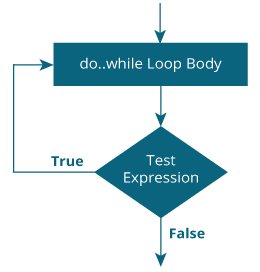
}

while (testExpression);

### How do...while loop works?

* The body of do...while loop is executed once. Only then, the testExpression is evaluated.
* If testExpression is **true**, the body of the loop is executed again and testExpression is evaluated once more.
* This process goes on until testExpression becomes **false**.
* If testExpression is **false**, the loop ends.

### Flowchart of do...while Loop

Working of do...while loop

### Example 2: do...while loop

// Program to add numbers until the user enters zero

#include <stdio.h>

int main() {

double number, sum = 0;

// the body of the loop is executed at least once

do {

printf("Enter a number: ");

scanf("%lf", &number);

sum += number;

}

while(number != 0.0);

printf("Sum = %.2lf",sum);

return 0;

}

**Output**

Enter a number: 1.5

Enter a number: 2.4

Enter a number: -3.4

Enter a number: 4.2

Enter a number: 0

Sum = 4.70

Here, we have used a do...while loop to prompt the user to enter a number. The loop works as long as the input number is not 0.

The do...while loop executes at least once i.e. the first iteration runs without checking the condition. The condition is checked only after the first iteration has been executed.

do {

printf("Enter a number: ");

scanf("%lf", &number);

sum += number;

}

while(number != 0.0);

So, if the first input is a non-zero number, that number is added to the sum variable and the loop continues to the next iteration. This process is repeated until the user enters 0.

But if the first input is 0, there will be no second iteration of the loop and sum becomes 0.0.

Outside the loop, we print the value of sum.

# C break and continue

We learned about loops in previous tutorials. In this tutorial, we will learn to use break and continue statements with the help of examples.

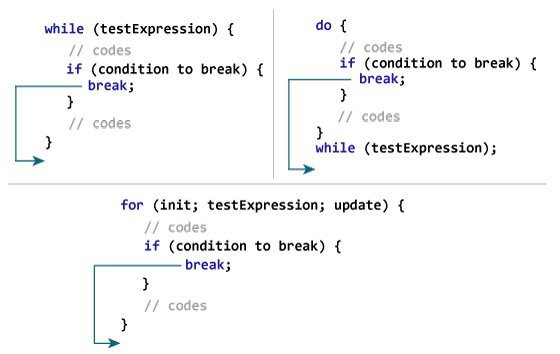
## C break

The break statement ends the loop immediately when it is encountered. Its syntax is:

break;

The break statement is almost always used with if...else statement inside the loop.

### How break statement works?

Working of break in C

### Example 1: break statement

// Program to calculate the sum of numbers (10 numbers max)

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

#include <stdio.h>

int main() {

int i;

double number, sum = 0.0;

for (i = 1; i <= 10; ++i) {

printf("Enter n%d: ", i);

scanf("%lf", &number);

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

if (number < 0.0) {

break;

}

sum += number; // sum = sum + number;

}

printf("Sum = %.2lf", sum);

return 0;

}

**Output**

Enter n1: 2.4

Enter n2: 4.5

Enter n3: 3.4

Enter n4: -3

Sum = 10.30

This program calculates the sum of a maximum of 10 numbers. Why a maximum of 10 numbers? It's because if the user enters a negative number, the break statement is executed. This will end the for loop, and the sum is displayed.

In C, break is also used with the switch statement. This will be discussed in the next tutorial.

## C continue

The continue statement skips the current iteration of the loop and continues with the next iteration. Its syntax is:

continue;

The continue statement is almost always used with the if...else statement.

### How continue statement works?

Working of Continue in C

### Example 2: continue statement

// Program to calculate the sum of numbers (10 numbers max)

// If the user enters a negative number, it's not added to the result

#include <stdio.h>

int main() {

int i;

double number, sum = 0.0;

for (i = 1; i <= 10; ++i) {

printf("Enter a n%d: ", i);

scanf("%lf", &number);

if (number < 0.0) {

continue;

}

sum += number; // sum = sum + number;

}

printf("Sum = %.2lf", sum);

return 0;

}

**Output**

Enter n1: 1.1

Enter n2: 2.2

Enter n3: 5.5

Enter n4: 4.4

Enter n5: -3.4

Enter n6: -45.5

Enter n7: 34.5

Enter n8: -4.2

Enter n9: -1000

Enter n10: 12

Sum = 59.70

In this program, when the user enters a positive number, the sum is calculated using sum += number; statement.

When the user enters a negative number, the continue statement is executed and it skips the negative number from the calculation.

# C switch Statement

In this tutorial, you will learn to create the switch statement in C programming with the help of an example.

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

You can do the same thing with the if...else..if ladder. However, the syntax of the switch statement is much easier to read and write.

## Syntax of switch...case

switch (expression)

​{

case constant1:

// statements

break;

case constant2:

// statements

break;

.

.

.

default:

// default statements

}

**How does the switch statement work?**

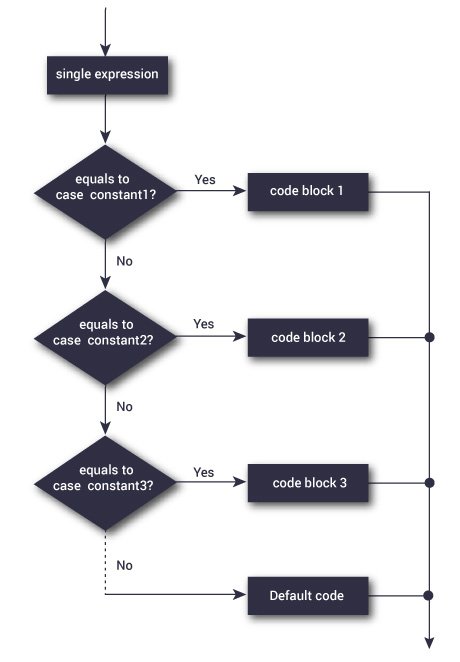
The expression is evaluated once and compared with the values of each case label.

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

**Notes:**

* If we do not use the break statement, all statements after the matching label are also executed.
* The default clause inside the switch statement is optional.

### switch Statement Flowchart

switch Statement Flowchart

### Example: Simple Calculator

// Program to create a simple calculator

#include <stdio.h>

int main() {

char operation;

double n1, n2;

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

scanf("%c", &operation);

printf("Enter two operands: ");

scanf("%lf %lf",&n1, &n2);

switch(operation)

{

case '+':

printf("%.1lf + %.1lf = %.1lf",n1, n2, n1+n2);

break;

case '-':

printf("%.1lf - %.1lf = %.1lf",n1, n2, n1-n2);

break;

case '\*':

printf("%.1lf \* %.1lf = %.1lf",n1, n2, n1\*n2);

break;

case '/':

printf("%.1lf / %.1lf = %.1lf",n1, n2, n1/n2);

break;

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

default:

printf("Error! operator is not correct");

}

return 0;

}

**Output**

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

Enter two operands: 32.5

12.4

32.5 - 12.4 = 20.1

The - operator entered by the user is stored in the operation variable. And, two operands 32.5 and 12.4 are stored in variables n1 and n2 respectively.

Since the operation is -, the control of the program jumps to

printf("%.1lf - %.1lf = %.1lf", n1, n2, n1-n2);

Finally, the [break statement](https://www.programiz.com/c-programming/c-break-continue-statement) terminates the switch statement.

# C goto Statement

In this tutorial, you will learn to create the goto statement in C programming. Also, you will learn when to use a goto statement and when not to use it.

The goto statement allows us to transfer control of the program to the specified label.

### Syntax of goto Statement

goto label;

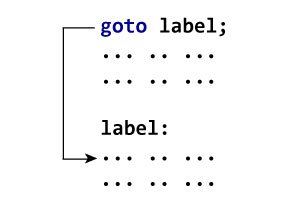
... .. ...

... .. ...

label:

statement;

The label is an identifier. When the goto statement is encountered, the control of the program jumps to label: and starts executing the code.

Working of goto Statement

### Example: goto Statement

// Program to calculate the sum and average of positive numbers

// If the user enters a negative number, the sum and average are displayed.

#include <stdio.h>

int main() {

const int maxInput = 100;

int i;

double number, average, sum = 0.0;

for (i = 1; i <= maxInput; ++i) {

printf("%d. Enter a number: ", i);

scanf("%lf", &number);

// go to jump if the user enters a negative number

if (number < 0.0) {

goto jump;

}

sum += number;

}

jump:

average = sum / (i - 1);

printf("Sum = %.2f\n", sum);

printf("Average = %.2f", average);

return 0;

}

**Output**

1. Enter a number: 3

2. Enter a number: 4.3

3. Enter a number: 9.3

4. Enter a number: -2.9

Sum = 16.60

Average = 5.53

### Reasons to avoid goto

The use of goto statement may lead to code that is buggy and hard to follow. For example,

one:

for (i = 0; i < number; ++i)

{

test += i;

goto two;

}

two:

if (test > 5) {

goto three;

}

... .. ...

Also, the goto statement allows you to do bad stuff such as jump out of the scope.

That being said, goto can be useful sometimes. For example: to break from nested loops.

### Should you use goto?

If you think the use of goto statement simplifies your program, you can use it. That being said, goto is rarely useful and you can create any C program without using goto altogether.

Here's a quote from Bjarne Stroustrup, creator of C++, **"The fact that 'goto' can do anything is exactly why we don't use it."**