

C if...else Statement

In programming, decision making is used to specify the order in which statements are executed. In this tutorial, you will learn to write if...else statements to make decisions in your program.

Table of Contents

- [if Statement](#)
 - [Example: if Statement](#)
- [if...else Statement](#)
 - [Example: if...else Statement](#)
- [if...else Ladder](#)
 - [Example: if...else Ladder](#)
- [Nested if...else](#)
 - [Example: Nested if...else](#)

C if statement

The syntax of `if` statement is:

```
if (testExpression)
{
    // statement(s)
}
```

How if statement works?

The `if` statement evaluates the test expression inside the parenthesis.


- If the test expression is evaluated to true (nonzero), statement(s) inside the body of `if` is executed.
- If the test expression is evaluated to false (0), statement(s) inside the body of `if` is skipped from execution.

Expression is true.

```
int test = 5;

if (test < 10)
{
    // codes
}

// codes after if
```

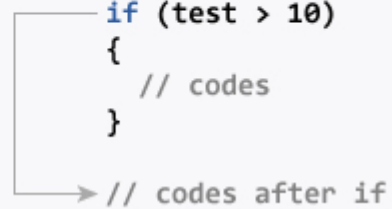


Expression is false.

```
int test = 5;

if (test > 10)
{
    // codes
}

// codes after if
```



To learn more on when test expression is evaluated to nonzero (true) and 0 (false), check [relational](#) and [logical operators](#).

Example 1: if statement

```
// Program to display a number if user enters negative number
#include <stdio.h>
int main()
{
    int number;

    printf("Enter an integer: ");
    scanf("%d", &number);

    // Test expression is 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 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 user enters 5, the test expression (`number < 0`) is evaluated to false and the statement inside the body of `if` is skipped.

C if...else statement

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

```
if (testExpression) {
    // statement(s) inside the body of if
}
else {
    // statement(s) inside the body of else
}
```

How if...else statement works?

If test expression is evaluated to true,

- statement(s) inside the body of `if` statement is executed
- statement(s) inside the body of `else` statement is skipped from execution.

If test expression is evaluated to false,

- statement(s) inside the body of `else` statement is executed
- statement(s) inside the body of `if` statement is skipped.

Expression is true.

```
int test = 5;

if (test < 10)
{
    // body of if
}
else
{
    // body of else
}
```

Expression is false.

```
int test = 5;

if (test > 10)
{
    // body of if
}
else
{
    // body of else
}
```

Example 2: if...else statement

```
// Program to check whether an integer entered by the user is odd or even
#include <stdio.h>
int main()
{
    int number;
    printf("Enter an integer: ");
    scanf("%d",&number);

    // True if 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 user enters 7, the test expression (`number%2 == 0`) is evaluated to false. Hence, the statement inside the body of `else` statement `printf("%d is an odd integer");` is executed and the statement inside the body of `if` is skipped.

if...else Ladder (if...else if...else Statement)

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 for multiple test expressions and execute different statement(s).

Syntax of nested `if...else` statement.

```
if (testExpression1)
{
    // statement(s)
}
else if(testExpression2)
{
    // statement(s)
}
else if (testExpression 3)
{
    // statement(s)
}
.
.
else
{
    // statement(s)
}
```

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

```
// Program to relate two integers using =, > or <
#include <stdio.h>
int main()
{
    int number1, number2;
    printf("Enter two integers: ");
    scanf("%d %d", &number1, &number2);

    //checks if two integers are equal.
    if(number1 == number2)
    {
        printf("Result: %d = %d",number1,number2);
    }

    //checks if number1 is greater than number2.
    if(number1 > number2)
    {
        printf("Result: %d > %d",number1,number2);
    }
}
```

```

else if (number1 > number2)
{
    printf("Result: %d > %d", number1, number2);
}

// if both test expression is 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 `if...else` statement(s) inside the body of another `if...else` statement.

This program below relates two integers using either `<`, `>` and `=` similar like in `if...else` ladder example. However, we will use nested `if...else` statement to solve this problem.

Example 4: Nested if...else

```

#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 `if...else` statement has only one statement, you do not need to use parenthesis `{ }`.

This code

```
if (a > b) {
    print("Hello");
}
print("Hi");
```

is equivalent to

```
if (a > b)
    print("Hello");
print("Hi");
```

Also Read: [C switch statement](#)

C Programming for Loop

Loops are used in programming to repeat a specific block of code. After reading this tutorial, you will learn to create for loop in C programming.

Table of Contents

- [What is loop?](#)
- [for loop \(and it's syntax\)](#)
- [How for loop works?](#)
- [for Loop flowchart](#)
- [Example: for loop](#)

Loops are used in programming to repeat a block of code until a specific condition is met. There are three loops in C programming:

1. for loop
2. [while loop](#)
3. [do...while loop](#)

for Loop

The syntax of for loop is:

```
for (initializationStatement; testExpression; updateStatement)
{
    // codes
}
```

How for loop works?

The initialization statement is executed only once.

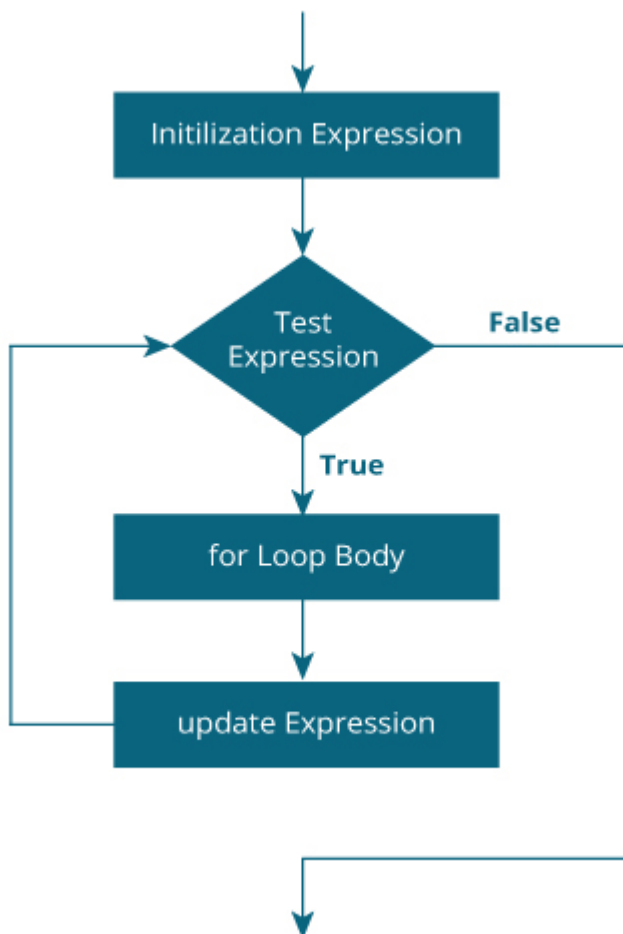
Then, the test expression is evaluated. If the test expression is false (0), for loop is terminated. But if the test expression is true (nonzero), codes inside the body of for loop is executed and the update expression is updated.

This process repeats until the test expression is false.

The `for` loop is commonly used when the number of iterations is known.

To learn more on test expression (when test expression is evaluated to nonzero (true) and 0 (false)), check out [relational](#) and [logical operators](#).

for loop Flowchart



Example: 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 n 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 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 for loop is executed. Now, the `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) as 11 is not less than or equal to 10. Therefore, the loop terminates and next, the total sum is printed.

C Programming while and do...while Loop

Loops are used in programming to repeat a specific block of code. After reading this tutorial, you will learn how to create while and do...while loop in C programming.

Table of Contents

- [What is loop?](#)
- [while Loop](#)
 - [How while loop works?](#)
 - [Flowchart of while loop](#)
 - [Example: while loop](#)
- [do...while Loop](#)
 - [How do...while loop works?](#)
 - [Flowchart of do...while loop](#)
 - [Example: do...while loop](#)

Loops are used in programming to repeat a block until a specific condition is met. There are three loops in C programming:

1. [for loop](#)
2. [while loop](#)
3. [do...while loop](#)

while loop

The syntax of a while loop is:

```
while (testExpression)
{
    //codes
}
```

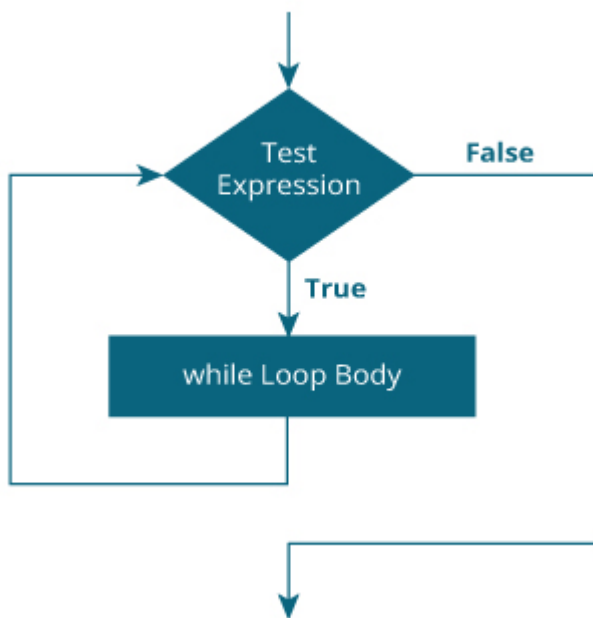
How while loop works?

The while loop evaluates the test expression.

If the test expression is true (nonzero), codes inside the body of while loop is executed. The test expression is evaluated again. The process goes on until the test expression is false.

When the test expression is false, the while loop is terminated.

Flowchart of while loop



Example 1: while loop

```
// Program to find factorial of a number
// For a positive integer n, factorial = 1*2*3...n

#include <stdio.h>
int main()
{
    int number;
    long long factorial;

    printf("Enter an integer: ");
    scanf("%d",&number);

    factorial = 1;

    // loop terminates when number is less than or equal to 0
    while (number > 0)
    {
        factorial *= number; // factorial = factorial*number;
        --number;
    }
}
```

```
printf("Factorial= %lld", factorial);  
  
return 0;  
}
```

Output

```
Enter an integer: 5  
Factorial = 120
```

To learn more on test expression (when test expression is evaluated to nonzero (true) and 0 (false)), check out [relational](#) and [logical operators](#).

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 once, before checking the test expression. Hence, the do...while loop is executed at least once.

do...while loop Syntax

```
do  
{  
    // codes  
}  
while (testExpression);
```

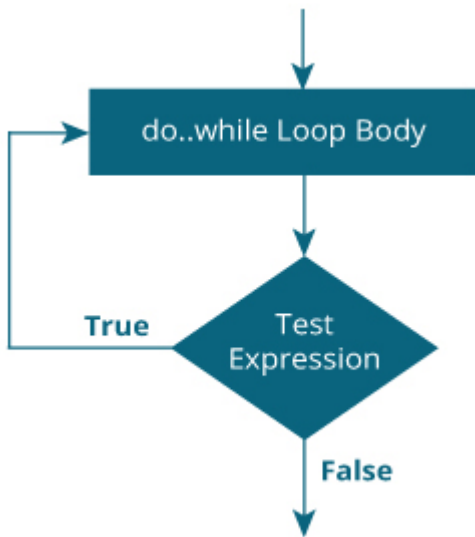
How do...while loop works?

The code block (loop body) inside the braces is executed once.

Then, the test expression is evaluated. If the test expression is true, the loop body is executed again. This process goes on until the test expression is evaluated to 0 (false).

When the test expression is false (nonzero), the do...while loop is terminated.

Flowchart of do...while Loop



Example 2: do...while loop

```
// Program to add numbers until user enters zero  
#include <stdio.h>  
int main()  
{  
    double number, sum = 0;  
  
    // body of 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
```

To learn more on test expression (when test expression is evaluated to nonzero (true) and 0 (false)), check out [relational](#) and [logical operators](#).

C Programming break and continue Statement

The break statement terminates the loop, whereas continue statement forces the next iteration of the loop. In this tutorial, you will learn to use break and continue with the help of examples.

Table of Contents

- [break statement](#)
 - [How break statement works?](#)
 - [Example: break statement](#)
- [continue statement](#)
 - [How continue statement works?](#)
 - [Example: continue statement](#)

It is sometimes desirable to skip some statements inside the loop or terminate the loop immediately without checking the test expression.

In such cases, break and continue statements are used.

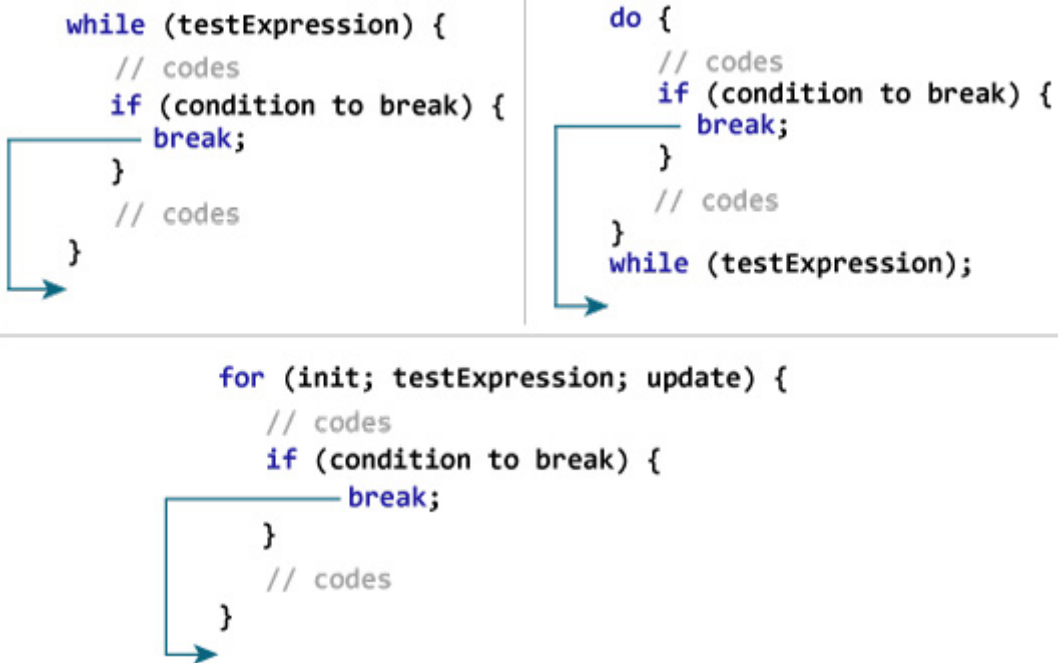
break Statement

The break statement terminates the loop ([for](#), [while](#) and [do...while 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?



Example 1: break statement

```

// Program to calculate the sum of maximum of 10 numbers
// If negative number is entered, loop terminates and sum is displayed

# 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 user enters negative number, loop is terminated
        if(number < 0.0)
        {
            break;
        }

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

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

    return 0;
}

```

Output

```
Enter a n1: 2.4
Enter a n2: 4.5
Enter a n3: 3.4
Enter a n4: -3
Sum = 10.30
```

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

In C, break is also used with [switch statement](#).

continue Statement

The continue statement skips statements after it inside the loop. Its syntax is:

```
continue;
```

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

How continue statement works?

```
while (testExpression) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

```
do {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
} while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

Example 2: continue statement

```
// Program to calculate sum of maximum of 10 numbers
// Negative numbers are skipped from calculation

# 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 a n1: 1.1
Enter a n2: 2.2
Enter a n3: 5.5
Enter a n4: 4.4
Enter a n5: -3.4
Enter a n6: -45.5
Enter a n7: 34.5
Enter a n8: -4.2
Enter a n9: -1000
Enter a n10: 12
Sum = 59.70
```

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

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

C switch...case Statement

In this tutorial, you will learn to write a switch statement in C programming (with an example).

Table of Contents

- [C switch statement](#)
- [Syntax of switch](#)
- [switch statement flowchart](#)
- [Example: switch statement](#)

The `if..else..if` ladder allows you to execute a block code among many alternatives. If you are checking on the value of a single variable in `if...else...if`, it is better to use `switch` statement.

The `switch` statement is often faster than nested `if...else` (not always). Also, the syntax of `switch` statement is cleaner and easy to understand.

Syntax of switch...case

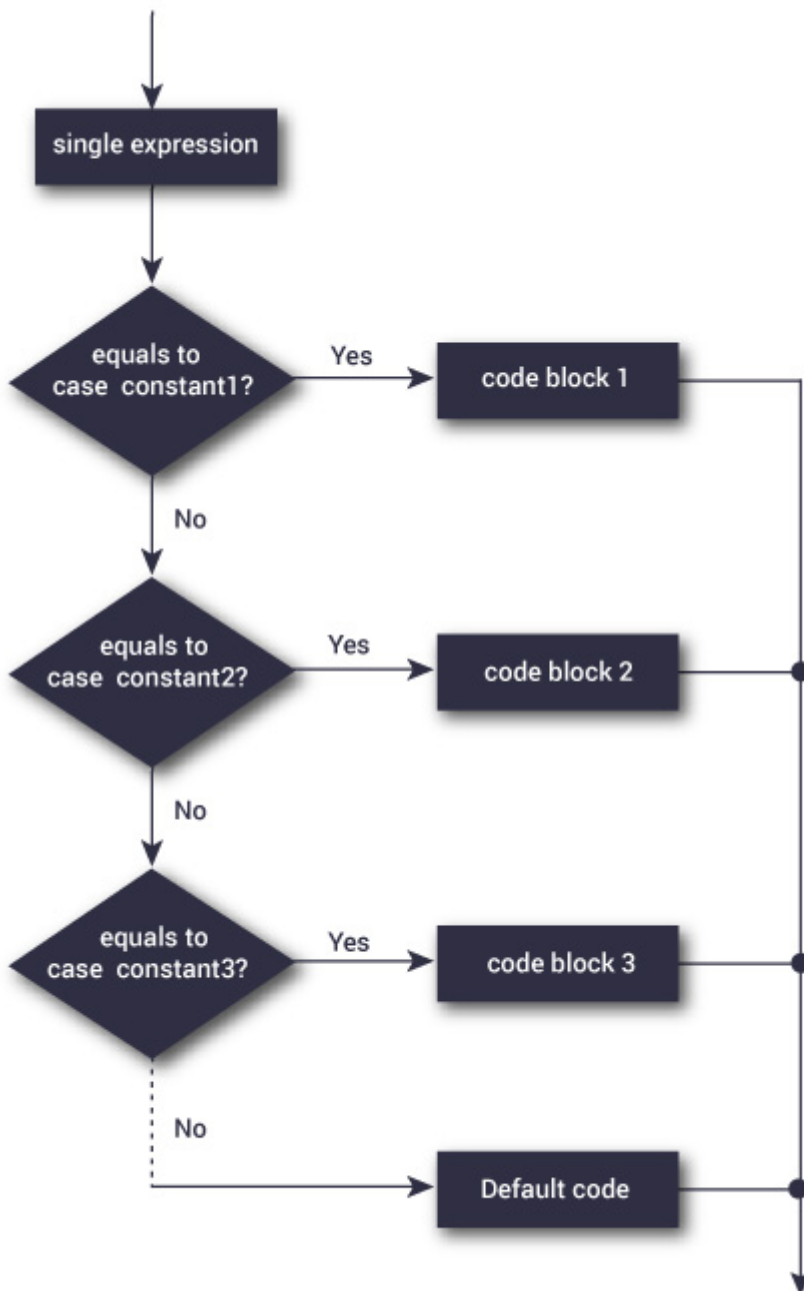
```
switch (n)
{
    case constant1:
        // code to be executed if n is equal to constant1;
        break;

    case constant2:
        // code to be executed if n is equal to constant2;
        break;
        .
        .
        .
    default:
        // code to be executed if n doesn't match any constant
}
```

When a case constant is found that matches the switch expression, control of the program passes to the block of code associated with that case.

Suppose, the value of `n` is equal to `constant2`. The compiler executes the statements after `case constant2:` until `break` is encountered. When `break` statement is encountered, switch statement terminates.

switch Statement Flowchart



Example: switch Statement

```

// Program to create a simple calculator
#include <stdio.h>

int main() {

    char operator;
    double firstNumber,secondNumber;

    printf("Enter an operator (+, -, *, /): ");
    scanf("%c", &operator);

    printf("Enter two operands: ");
    scanf("%lf %lf",&firstNumber, &secondNumber);

    switch(operator)
    {
        case '+':
            printf("%.1lf + %.1lf = %.1lf",firstNumber, secondNumber,
            break;

        case '-':
            printf("%.1lf - %.1lf = %.1lf",firstNumber, secondNumber,
            break;

        case '*':
            printf("%.1lf * %.1lf = %.1lf",firstNumber, secondNumber,
            break;

        case '/':
            printf("%.1lf / %.1lf = %.1lf",firstNumber, secondNumber,
            break;

        // operator doesn't match any case constant (+, -, *, /)
        default:
            printf("Error! operator is not correct");
    }
}

```

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 operator variable. And, two operands 32.5 and 12.4 are stored in variables firstNumber and secondNumber respectively.

Then, control of the program jumps to

```

printf("%.1lf / %.1lf = %.1lf",firstNumber, secondNumber, firstNumber/first

```

Finally, the `break statement` terminates the switch statement.

C goto Statement

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

Table of Contents

- [Syntax of goto](#)
- [Example: goto statement](#)
- [Reasons to avoid goto](#)
- [Should you use goto statement?](#)



The goto statement is used to alter the normal sequence of a C program.

Syntax of goto statement


```
goto label;
... ..
... ..
... ..
label:
statement;
```

The label is an [identifier](#). When `goto` statement is encountered, control of the program jumps to `label:` and starts executing the code.



Example: goto Statement

```
// Program to calculate the sum and average of maximum of 5 numbers
// If user enters negative number, the sum and average of previously entered numbers are reset
#include <stdio.h>

int main()
{
    const int maxInput = 5;
    int i;
    double number, average, sum=0.0;

    for(i=1; i<=maxInput; ++i)
    {
        printf("%d. Enter a number: ", i);
        scanf("%lf",&number);

        // If user enters negative number, flow of program moves to label jump
        if(number < 0.0)
            goto jump;

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

Reasons to avoid goto statement

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

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

Should you use goto statement?

If you think the use of goto statement simplifies your program. By all means use it. The goal here is to create code that your fellow programmers can understand easily.

