**break, continue, switch..case Statement**

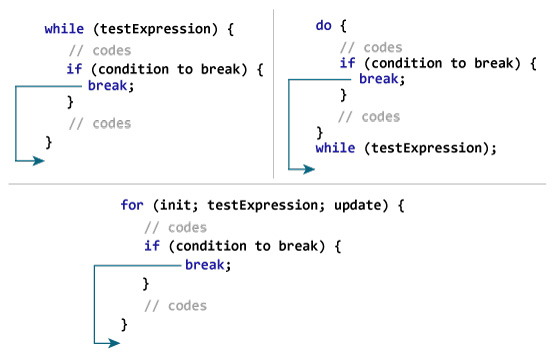
**break** :

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?**



**Example 1: break statement**

// Program to calculate the sum of a maximum of 10 numbers

// If a negative number is entered, the loop terminates

# 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 the user enters a negative number, the loop ends

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 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.

## Example using break

The following function, trim, removes trailing blanks, tabs and newlines from the end of a string, using a break to exit from a loop when the rightmost non-blank, non-tab, non-newline is found.

*/\* trim: remove trailing blanks, tabs, newlines \*/*

int trim(char s[])

{

int n;

for (n = [strlen](http://www.opengroup.org/onlinepubs/009695399/functions/strlen.html)(s)-1; n >= 0; n--)

if (s[n] != ' ' && s[n] != '**\t**' && s[n] != '**\n**')

**break**;

s[n+1] = '**\0**';

return n;

}

strlen returns the length of the string. The for loop starts at the end and scans backwards looking for the first character that is not a blank or tab or newline. The loop is broken when one is found, or when n becomes negative (that is, when the entire string has been scanned).

In C, break is also used with the switch statement.

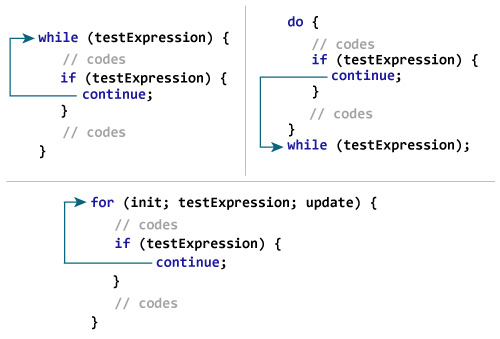
**continue:**

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?**



**Example 2: continue statement**

// Program to calculate the sum of a maximum of 10 numbers

// Negative numbers are skipped from the 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 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.

## Example using continue

As an example, the following piece of code sums up the non-negative elements in the array a; negative values are skipped.

*/\* sum up non-negative elements of an array \*/*

#include <stdio.h>

int main()

{

int a[10] = {-1, 2, -3, 4, -5, 6, -7, 8, -9, 10};

int i, sum = 0;

for (i = 0; i < 10; i++)

{

if (a[i] < 0) */\* skip negative elements \*/*

continue;

sum += a[i]; */\* sum positive elements \*/*

}

[printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Sum of positive elements: %d**\n**", sum);

}

OUTPUT

======

Sum of positive elements: 30

|  |  |
| --- | --- |
| Difference Between break and continue | |
| **break** | **continue** |
| A break can appear in both switch and loop (for, while, do) statements. | A continue can appear only in loop (for, while, do) statements. |
| A break causes the switch or loop statements to terminate the moment it is executed. Loop or switch ends abruptly when break is encountered. | A continue doesn't terminate the loop, it causes the loop to go to the next iteration. All iterations of the loop are executed even if continue is encountered. The continue statement is used to skip statements in the loop that appear after the continue. |
| The break statement can be used in both switch and loop statements. | The continue statement can appear only in loops. You will get an error if this appears in switch statement. |
| When a break statement is encountered, it terminates the block and gets the control out of the switch or loop. | When a continue statement is encountered, it gets the control to the next iteration of the loop. |
| A break causes the innermost enclosing loop or switch to be exited immediately. | A continue inside a loop nested within a switch causes the next loop iteration. |

**Similarities Between break and continue**

Both break and continue statements in C programming language have been provided to alter the normal flow of program.

## switch...case Statement

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

A switch statement tests the value of a variable and compares it with multiple cases. Once the case match is found, a block of statements associated with that particular case is executed.

Each case in a block of a switch has a different name/number which is referred to as an identifier. The value provided by the user is compared with all the cases inside the switch block until the match is found.

If a case match is found, then the default statement is executed, and the control goes out of the switch block.

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?**

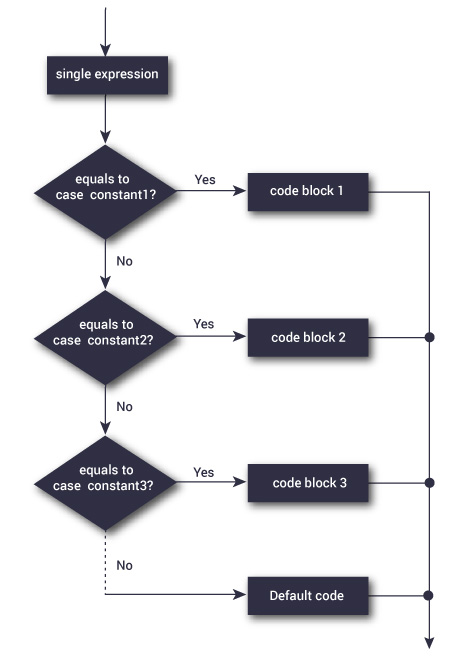
The expression is evaluated once and compared with the values of each case label.

* 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.

If we do not use break, all statements after the matching label are executed.

By the way, the default clause inside the switch statement is optional.

### switch Statement Flowchart



### Example: Simple Calculator

// Program to create a simple calculator

#include <stdio.h>

int main() {

char operator;

double n1, n2;

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

scanf("%c", &operator);

printf("Enter two operands: ");

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

switch(operator)

{

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 *operator* variable. And, two operands *32.5* and *12.4* are stored in variables *n1* and *n2* respectively.

Since the *operator* 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.

Example:

#include <stdio.h>

int main()

{

int num = 8;

switch (num) {

case 7:

printf("Value is 7");

break;

case 8:

printf("Value is 8");

break;

case 9:

printf("Value is 9");

break;

default:

printf("Out of range");

break;

}

return 0;

}

Output:

Value is 8