C break and continue

Break statement:

The **break** statement in C programming has the following two usages −

- When a **break** statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
- It can be used to terminate a case in the **switch** statement.

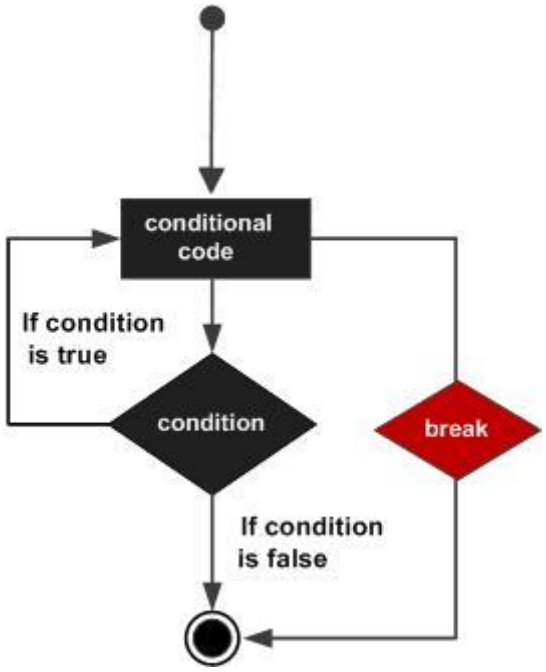
If you are using nested loops, the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.

Syntax

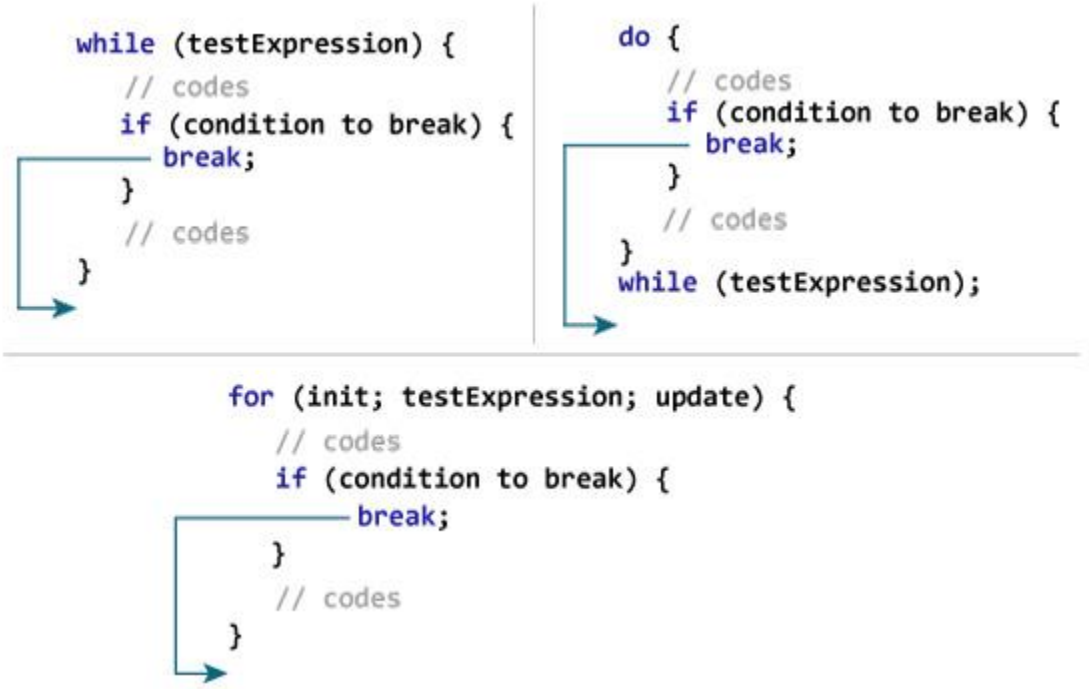
The syntax for a **break** statement in C is as follows −

```
break;
```

Flow Diagram



How break statement works?



Example: 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.

Continue statement:

The **continue** statement in C programming works somewhat like the **break** statement. Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between.

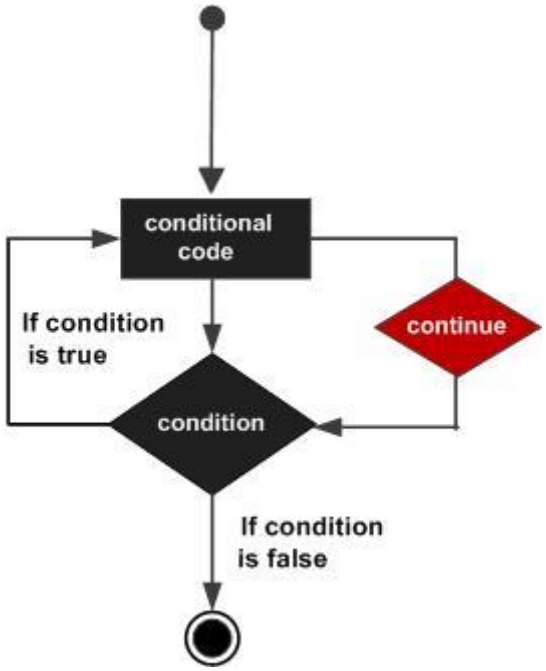
For the **for** loop, **continue** statement causes the conditional test and increment portions of the loop to execute. For the **while** and **do...while** loops, **continue** statement causes the program control to pass to the conditional tests.

Syntax

The syntax for a **continue** statement in C is as follows −

```
continue;
```

Flow Diagram



How continue statement works?

Example-1: skip the iteration

```
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* do loop execution */
   do {

      if( a == 15) {
         /* skip the iteration */
         a = a + 1;
         continue;
      }

      printf("value of a: %d\n", a);
      a++;

   } while( a < 20 );

   return 0;
}
```

Output

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
```

```
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Example 2: 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.

Goto statement

A **goto** statement in C programming provides an unconditional jump from the 'goto' to a labeled statement in the same function.

NOTE − Use of **goto** statement is highly discouraged in any programming language because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a goto can be rewritten to avoid them.

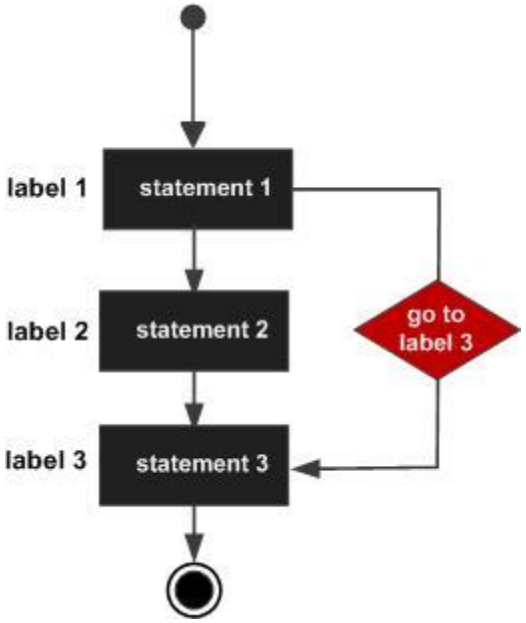
Syntax

The syntax for a **goto** statement in C is as follows −

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

Here **label** can be any plain text except C keyword and it can be set anywhere in the C program above or below to **goto** statement.

Flow Diagram



Example 1: skip the iteration

```
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* do loop execution */
   LOOP:do {

      if( a == 15) {
         /* skip the iteration */
         a = a + 1;
```

```
         goto LOOP;
      }

      printf("value of a: %d\n", a);
      a++;

   }while( a < 20 );

   return 0;
}
```

Output

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Example 2: 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 = 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(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: -1.2

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,

```
1. one:
2. for (i = 0; i < number; ++i)
3. {
4.     test += i;
5.     goto two;
6. }
7. two:
8. if (test > 5) {
9.   goto three;
10.  }
11.  ... .. ...
```

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.