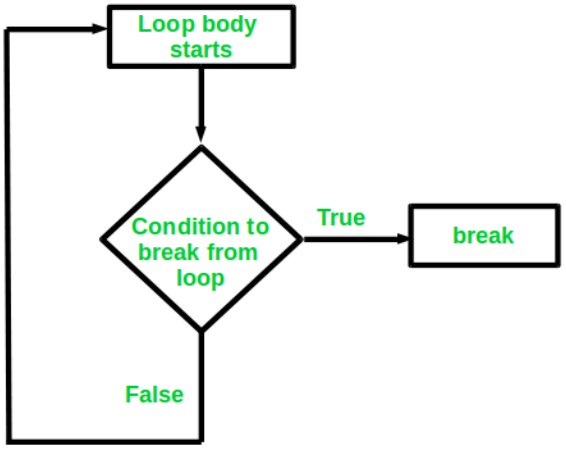
# Break in C++

The break in  C++ is a loop control statement which is used to terminate the loop. As soon as the break statement is encountered from within a loop, the loop iterations stops there and control returns from the loop immediately to the first statement after the loop.

**Syntax:**

```
break;
```

Basically break statements are used in the situations when we are not sure about the actual number of iterations for the loop or we want to terminate the loop based on some condition.




**We will see here the usage of break statement with three different types of loops:**

1. Simple loops
2. Nested loops
3. Infinite loops

Let us now look at the examples for each of the above three types of loops using break statement.

1. **Simple loops**: Consider the situation where we want to search an element in an array. To do this, use a loop to traverse the array starting from the first index and compare the array elements with the given key. Below is the implementation of this idea:

```
// CPP program to illustrate
// Linear Search
```

```
#include <iostream>
using namespace std;

void findElement(int arr[], int size, int key)
{
    // loop to traverse array and search for key
    for (int i = 0; i < size; i++) {
        if (arr[i] == key) {
            cout << "Element found at position: " << (i + 1);
        }
    }
}

// Driver program to test above function
int main()
{
    int arr[] = { 1, 2, 3, 4, 5, 6 };
    int n = 6; // no of elements
    int key = 3; // key to be searched

    // Calling function to find the key
    findElement(arr, n, key);

    return 0;
}
```

**Output**

```
Element found at position: 3
```

The above code runs fine with no errors. But the above code is not efficient. The above code completes all the iterations even after the element is found. Suppose there are **1000** elements in the array and the key to be searched is present at 1st position so the above approach will execute **999** iterations which are of no purpose and are useless.

To avoid these useless iterations, we can use the break statement in our program. Once the break statement is encountered the control from the loop will return immediately after the condition gets satisfied. So will use the break statement with the if condition which compares the key with array elements as shown below:

```
// CPP program to illustrate
// using break statement
// in Linear Search
#include <iostream>
using namespace std;

void findElement(int arr[], int size, int key)
{
    // loop to traverse array and search for key
    for (int i = 0; i < size; i++) {
        if (arr[i] == key) {
            cout << "Element found at position: " << (i + 1);

            // using break to terminate loop execution
            break;
        }
    }
}

// Driver program to test above function
int main()
{
    int arr[] = { 1, 2, 3, 4, 5, 6 };
    int n = 6; // no of elements
    int key = 3; // key to be searched

    // Calling function to find the key
    findElement(arr, n, key);

    return 0;
}
```

**Output**

```
Element found at position: 3
```

**Nested Loops**: We can also use break statement while working with nested loops. If the break statement is used in the innermost loop. The control will come out only from the innermost loop. Below is the example of using break with nested loops:

```
// CPP program to illustrate
// using break statement
// in Nested loops
#include <iostream>
using namespace std;

int main()
{

    // nested for loops with break statement
    // at inner loop
    for (int i = 0; i < 5; i++) {
        for (int j = 1; j <= 10; j++) {
            if (j > 3)
                break;
            else
                cout << "*";
        }
        cout << endl;
    }

    return 0;
}
```

**Output**

```
***
***
***
***
***
```

In the above code we can clearly see that the inner loop is programmed to execute for 10 iterations. But as soon as the value of **j** becomes greater than 3 the inner loop stops executing

which restricts the number of iteration of the inner loop to 3 iterations only. However the iteration of outer loop remains unaffected.

**Therefore, break applies to only the loop within which it is present.**

**Infinite Loops**: break statement can be included in an infinite loop with a condition in order to terminate the execution of the infinite loop.

Consider the below infinite loop:

```
// CPP program to illustrate
// using break statement
// in Infinite loops
#include <iostream>
using namespace std;

int main()
{
    // loop initialization expression
    int i = 0;

    // infinite while loop
    while (1) {
        cout << i << " ";
        i++;
    }

    return 0;
}
```

NOTE : Please do not run the above program in your compiler as it is an infinite loop so you may have to forcefully exit the compiler to terminate the program.

In the above program, the loop condition based on which the loop terminates is always true. So, the loop executes infinite number of times. We can correct this by using the break statement as shown below:

```
// CPP program to illustrate
// using break statement
// in Infinite loops
#include <iostream>
using namespace std;

int main()
{
    // loop initialization expression
    int i = 1;

    // infinite while loop
    while (1) {
        if (i > 10)
            break;

        cout << i << " ";
        i++;
    }

    return 0;
}
```

**Output**

```
1 2 3 4 5 6 7 8 9 10
```

Apart from this, break can be used in Switch case statements too.