**While loop syntax:**

The syntax of a while loop in C++ is:

```cpp
while (condition is true) {
    // statement(s);
}
```

Here, `statement(s)` may be a single statement or a block of statements. The `condition` may be any expression, and true is any non-zero value. The loop iterates while the condition is true.

When the condition becomes false, program control passes to the line immediately following the loop.

Beware of an infinite loop, don’t create conditions that are always true, unless you know what you are doing and it is intended:

```cpp
while (true) {
    // will run forever
}
```

Here, key point of the `while` loop is that the loop might not ever run. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the `while` loop will be executed.
Example:

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

int main () {
    // Local variable declaration:
    int a = 10;
    // while loop execution
    while (a < 20) {
        cout << "value of a: " << a;
        a++;  // the shortcut for a = a + 1
    }
    return 0;
}
```

When the above code is compiled and executed, it will print the countdown from 10 to 19:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```
**Logical operators (\!, \&\&, \|\|)\**

The operator ! is the C++ operator for the Boolean operation NOT. It has only one operand, to its right, and inverts it, producing false if its operand is true, and true if its operand is false. Basically, it returns the opposite Boolean value of evaluating its operand. For example:

```cpp
1  !(5 == 5)  // evaluates to false because the expression at its right (5 == 5) is true
2  !(6 <= 4)  // evaluates to true because (6 <= 4) would be false
3  !true      // evaluates to false
4  !false     // evaluates to true
```

The logical operators \&\& and || are used when evaluating two expressions to obtain a single relational result. The operator \&\& corresponds to the Boolean logical operation **AND**, which yields true if both its operands are true, and false otherwise. The following panel shows the result of operator \&\& evaluating the expression a\&\&b:

<table>
<thead>
<tr>
<th>&amp; &amp; OPERATOR (and)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>true</td>
</tr>
<tr>
<td>true</td>
</tr>
<tr>
<td>false</td>
</tr>
<tr>
<td>false</td>
</tr>
</tbody>
</table>
The operator || corresponds to the Boolean logical operation OR, which yields true if either of its operands is true, thus being false only when both operands are false. Here are the possible results of a||b:

|| OPERATOR (or)
| a | b | a || b |
|---|---|------|
| true | true | true |
| true | false | true |
| false | true | true |
| false | false | false |

For example:

1. `(5 == 5) && (3 > 6)` // evaluates to false (true && false)
2. `(5 == 5) || (3 > 6)` // evaluates to true (true || false)

When using the logical operators, C++ only evaluates what is necessary from left to right to come up with the combined relational result, ignoring the rest. Therefore, in the last example ((5==5)||(3>6)), C++ evaluates first whether 5==5 is true, and if so, it never checks whether 3>6 is true or not. This is known as short-circuit evaluation.