C++, like many other languages has its operations precedence, conditional statements, boolean and short-circuiting.

Booleans

Its boolean values are 1 and 0 with a the true statement outputting a 1 and the false statement outputting a 0. Take for example,

std:: cout << (7>8); The result would be false, the output would be 0.

Conditional statements

The conditional statements include logical conditions that are well known in mathematics with the comparison values such as >,<,== and so on. The conditions C++ allows are the usual if/else statements. It also allows else if statements which would go in between the if and the else. Else if statements are C++’s version of elif statements, an alternative solution would would be to place if else statements within themselves. An example being starting with an if, following with an else, but in that else having another if followed by an else. However, used incorrectly, that version can get confusing so the safest straight forward option would be to utlitize the else if. Lastly, C++ utilizes switch statements. Now switch statements are useful when multiple blocks of code go in a specific statement. These statements come with an option to use “break” in order to stop running through that specific subprogram and output what was sought. Using “break” is more of a preference/ convenience of the specific program being coded so think carefully if you want to stop running through your options then and there, or if the following additional cases are useful and have a purpose to the output you are seeking.

<https://www.w3schools.com/cpp/cpp_conditions.asp>

Operators Precedence

When it comes to Operators Precedence, operators with a higher precedence have priority so that the values and operations in an expression are not all evaluated at once but rather the ones with higher precedence are evaluated first. There are multiple tables available with the precedence, operator, description and associativity that can be found. Two of which the links will be provided.

<https://www.programiz.com/cpp-programming/operators-precedence-associativity>

<https://en.cppreference.com/w/cpp/language/operator_precedence>

Delimiter

C++ has one delimiter for statements, the semicolon. The use of the semicolon is what the program recognizes as the end of a statement rather than the end of the code itself. For this reason, indentation is not a priority so multiple statements can be on the same line as long as each statement is followed by the semicolon. For example rather than having 3 lines, each representing, int a = 5; int b = 6; and cout << a + b; one line can can accomplish the same → int a = 5; int b = 6; std :: cout << a + b;

<https://www.softwaretestinghelp.com/cpp-basics/>

In the case of code blocks, the curly brackets are typically what group together the code for conditions. That being said an if statement without brackets is permitted but it will be declared to only have one line of code as a result. This poses a concern known as the “dangling else” . The “dangling else” problem is a syntatic ambiguity issue that results with nested ifs. If statements can easily confuse the else statement, leaving the else statement wondering which if statement it belongs to.

<https://www.geeksforgeeks.org/dangling-else-ambiguity/>

Short-Circuit

C++ does use short-circuit logic. Short-circuiting allows the program to quickly work through the code without having to pass through unnecessary expressions and conditions when the beginning of the expression automatically declares the result of the entire expression. It takes a short-cut getting to the output as soon as possible.