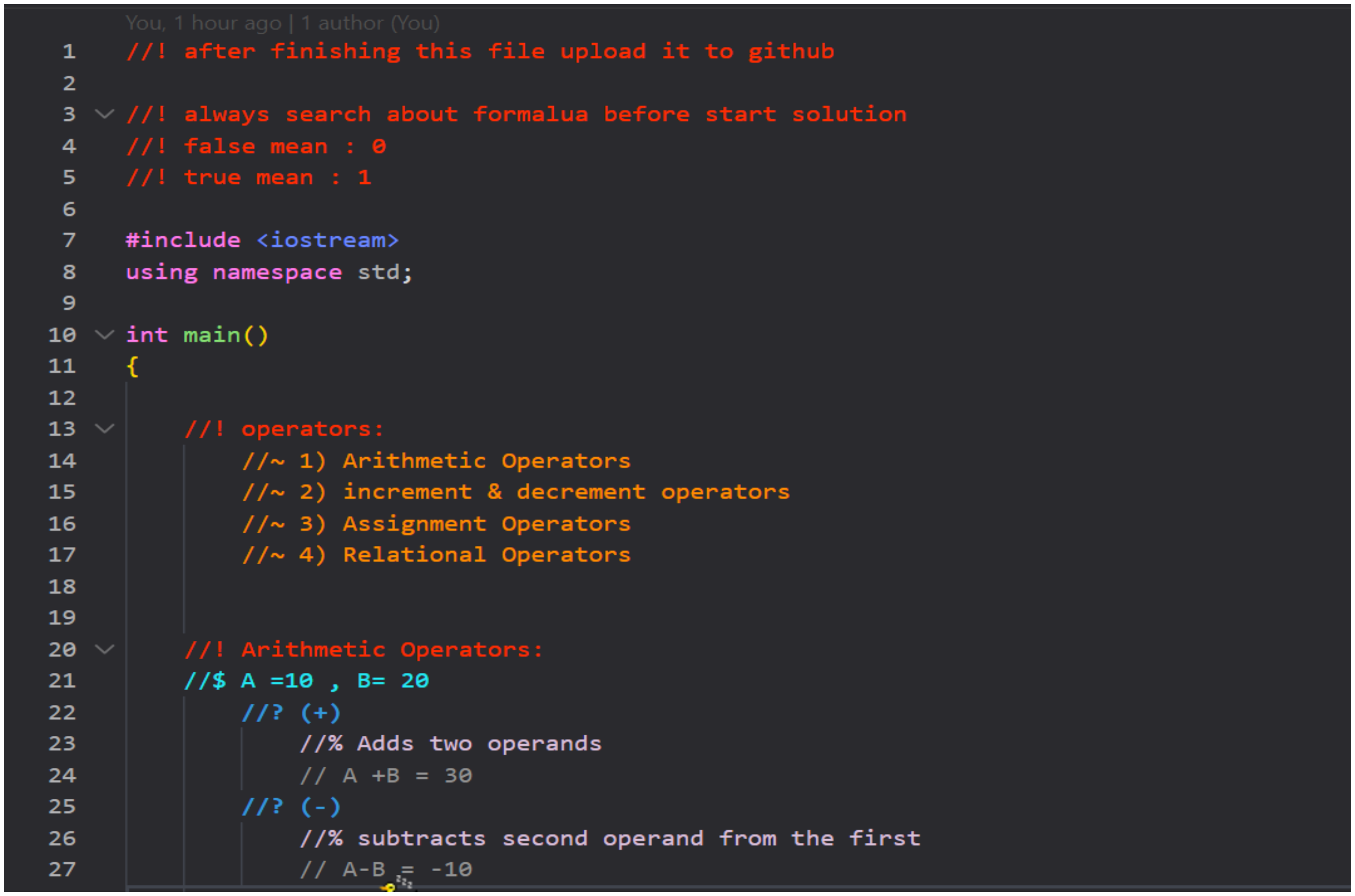
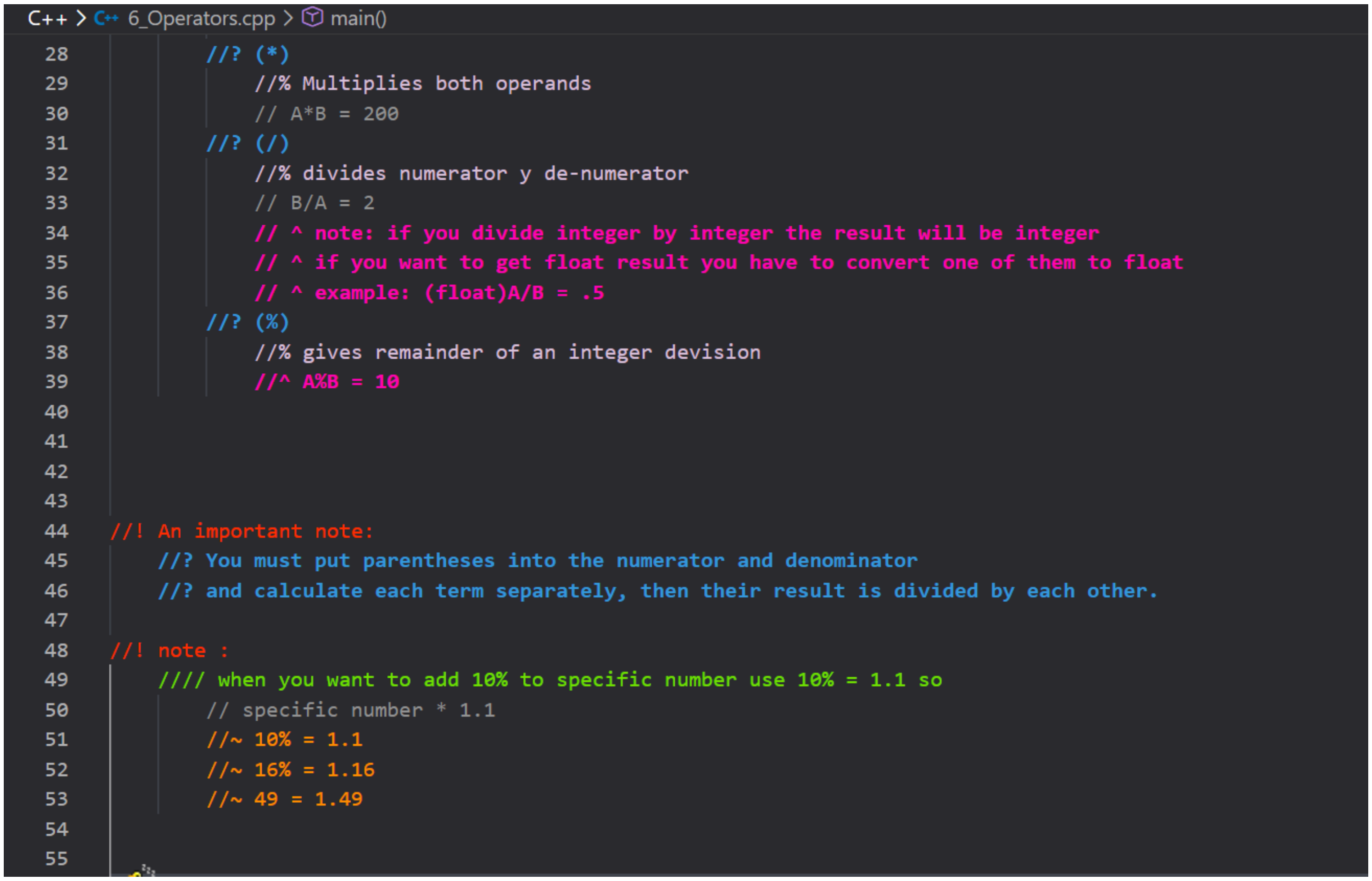
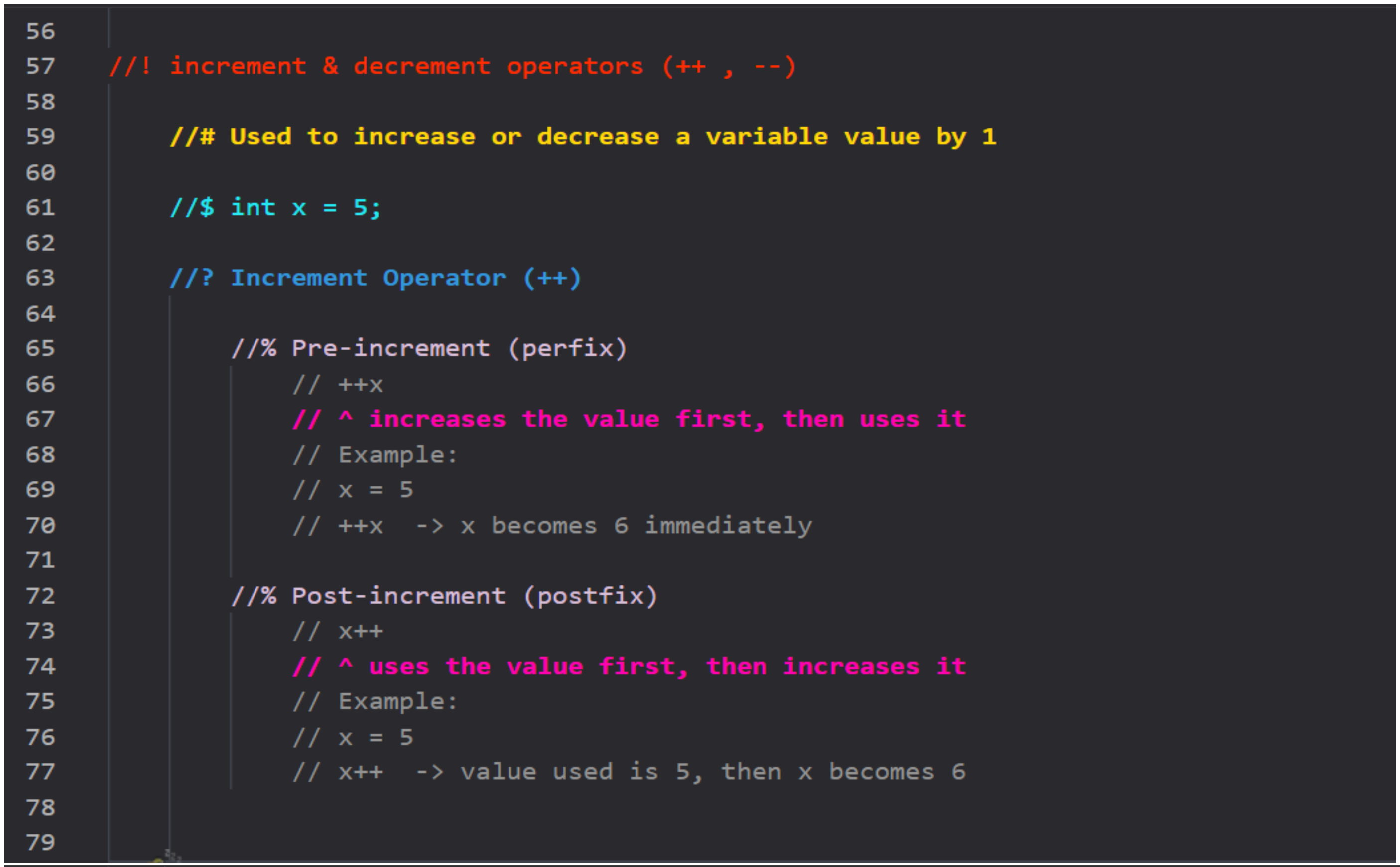

You, 1 hour ago | 1 author (You)

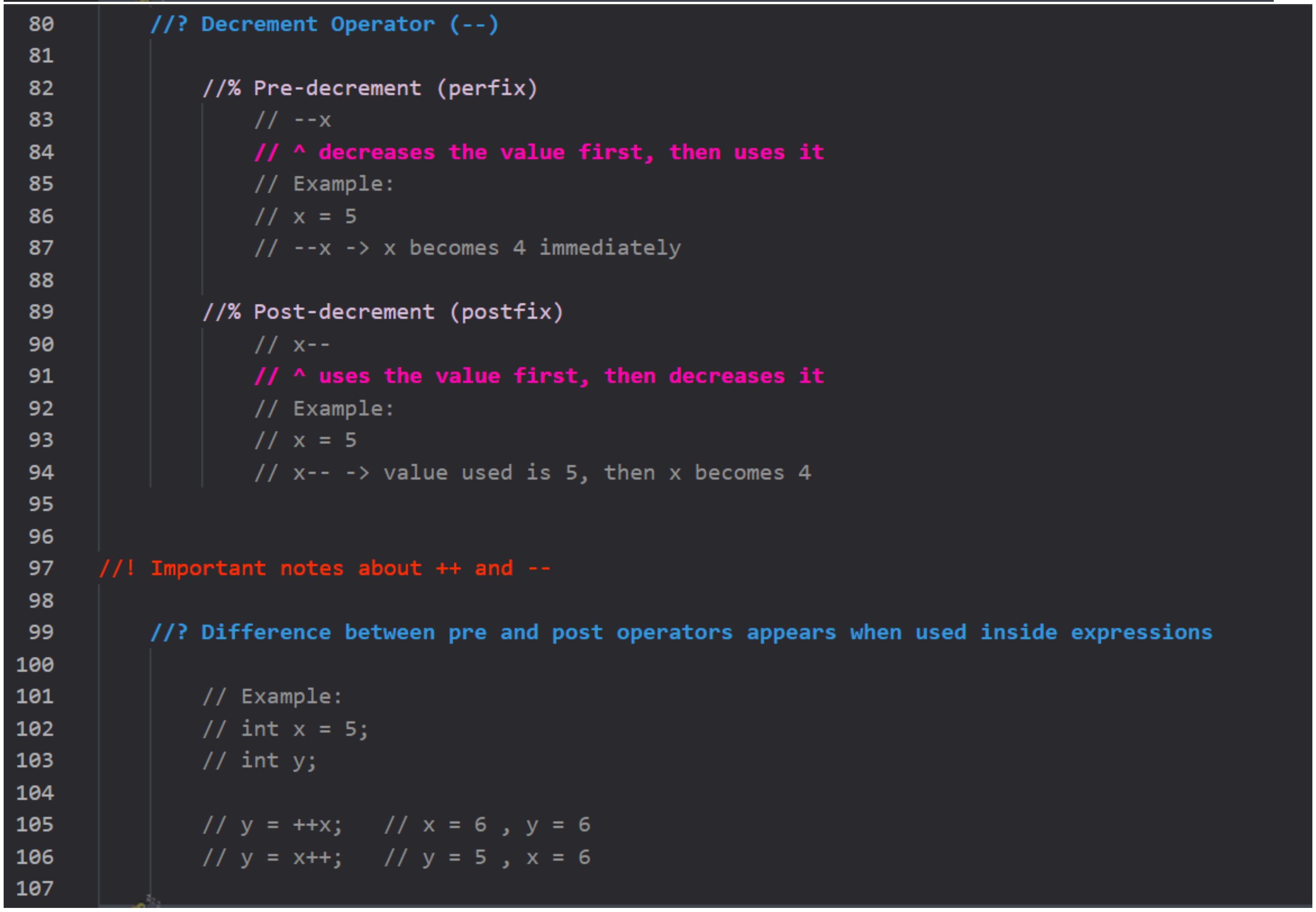
```cpp
//! after finishing this file upload it to github

//! always search about formalua before start solution
//! false mean : 0
//! true mean : 1

#include <iostream>
using namespace std;

int main()
{

    //! operators:
        //~ 1) Arithmetic Operators
        //~ 2) increment & decrement operators
        //~ 3) Assignment Operators
        //~ 4) Relational Operators


    //! Arithmetic Operators:
    //$ A =10 , B= 20
        //? (+)
            //% Adds two operands
            // A +B = 30
        //? (-)
            //% subtracts second operand from the first
            // A-B = -10
```


C++ > 6_Operators.cpp > main()

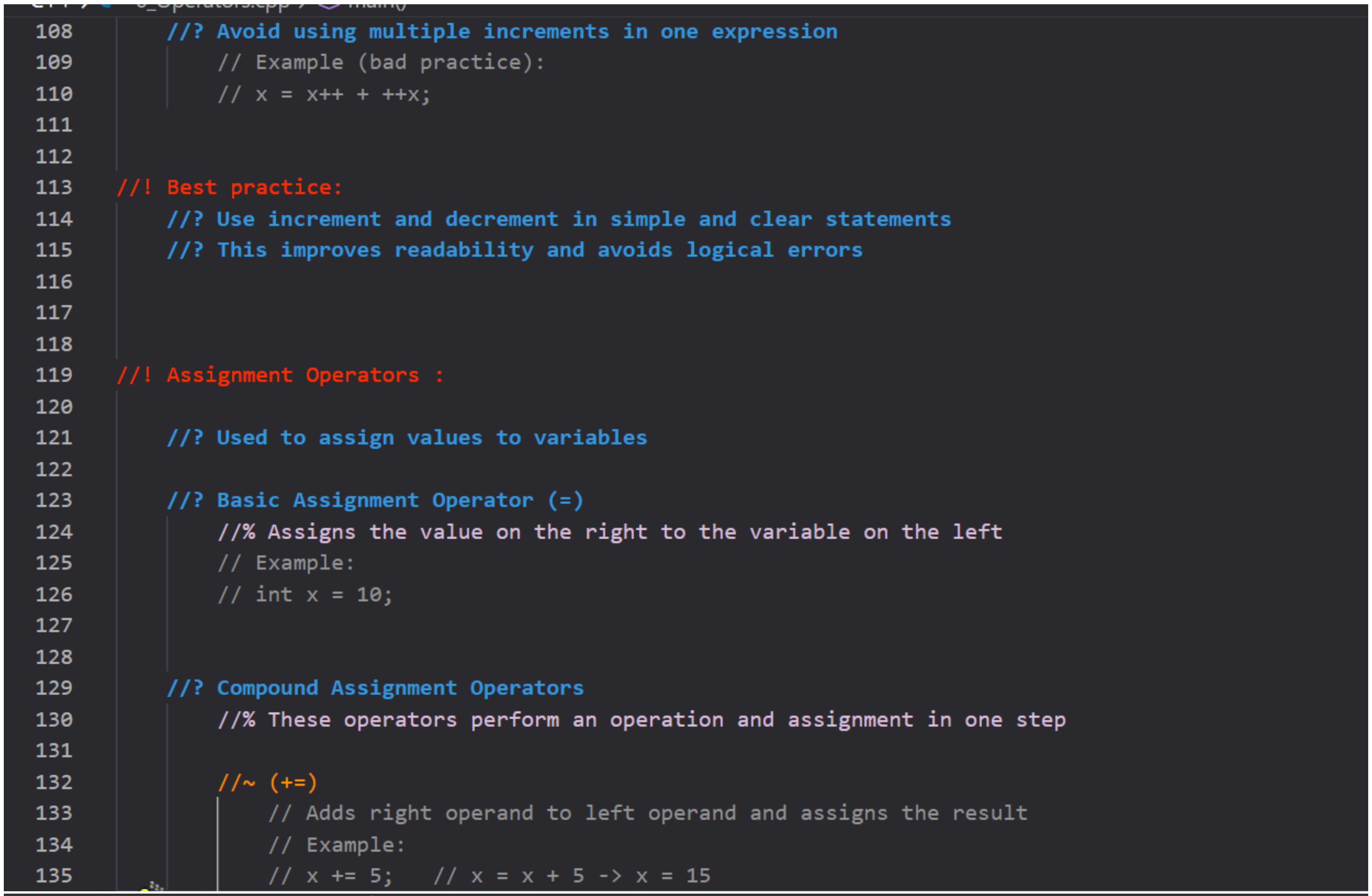
```cpp
        //? (*)
            //% Multiplies both operands
            // A*B = 200
        //? (/)
            //% divides numerator y de-numerator
            // B/A = 2
            // ^ note: if you divide integer by integer the result will be integer
            // ^ if you want to get float result you have to convert one of them to float
            // ^ example: (float)A/B = .5
        //? (%)
            //% gives remainder of an integer devision
            //^ A%B = 10




//! An important note:
    //? You must put parentheses into the numerator and denominator
    //? and calculate each term separately, then their result is divided by each other.

//! note :
    //// when you want to add 10% to specific number use 10% = 1.1 so
        // specific number * 1.1
        //~ 10% = 1.1
        //~ 16% = 1.16
        //~ 49 = 1.49

```



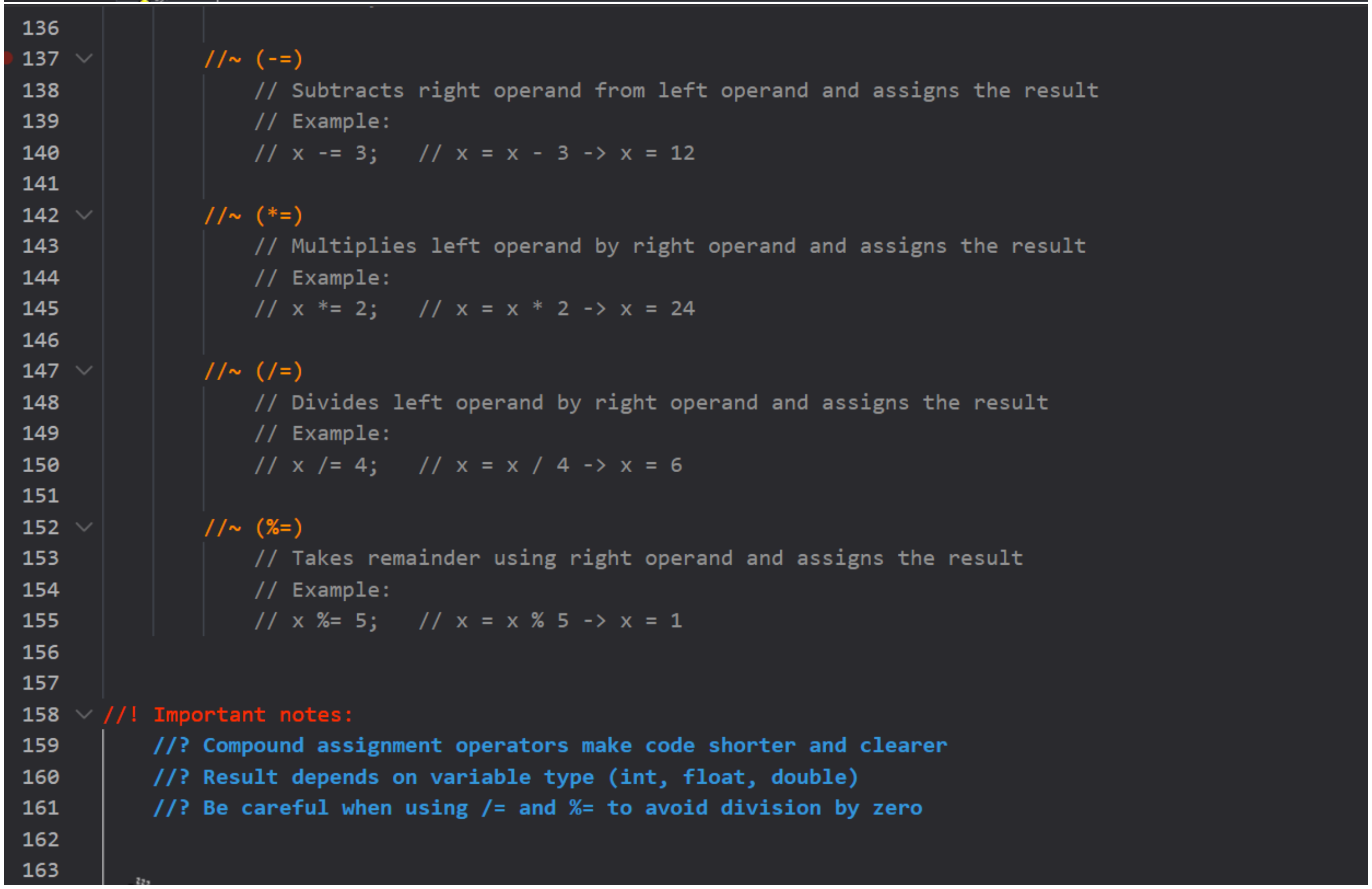
```
//! increment & decrement operators (++ , --)

    //# Used to increase or decrease a variable value by 1

    //$ int x = 5;

    //? Increment Operator (++)

        //% Pre-increment (perfix)
            // ++x
            // ^ increases the value first, then uses it
            // Example:
            // x = 5
            // ++x  -> x becomes 6 immediately

        //% Post-increment (postfix)
            // x++
            // ^ uses the value first, then increases it
            // Example:
            // x = 5
            // x++  -> value used is 5, then x becomes 6


```



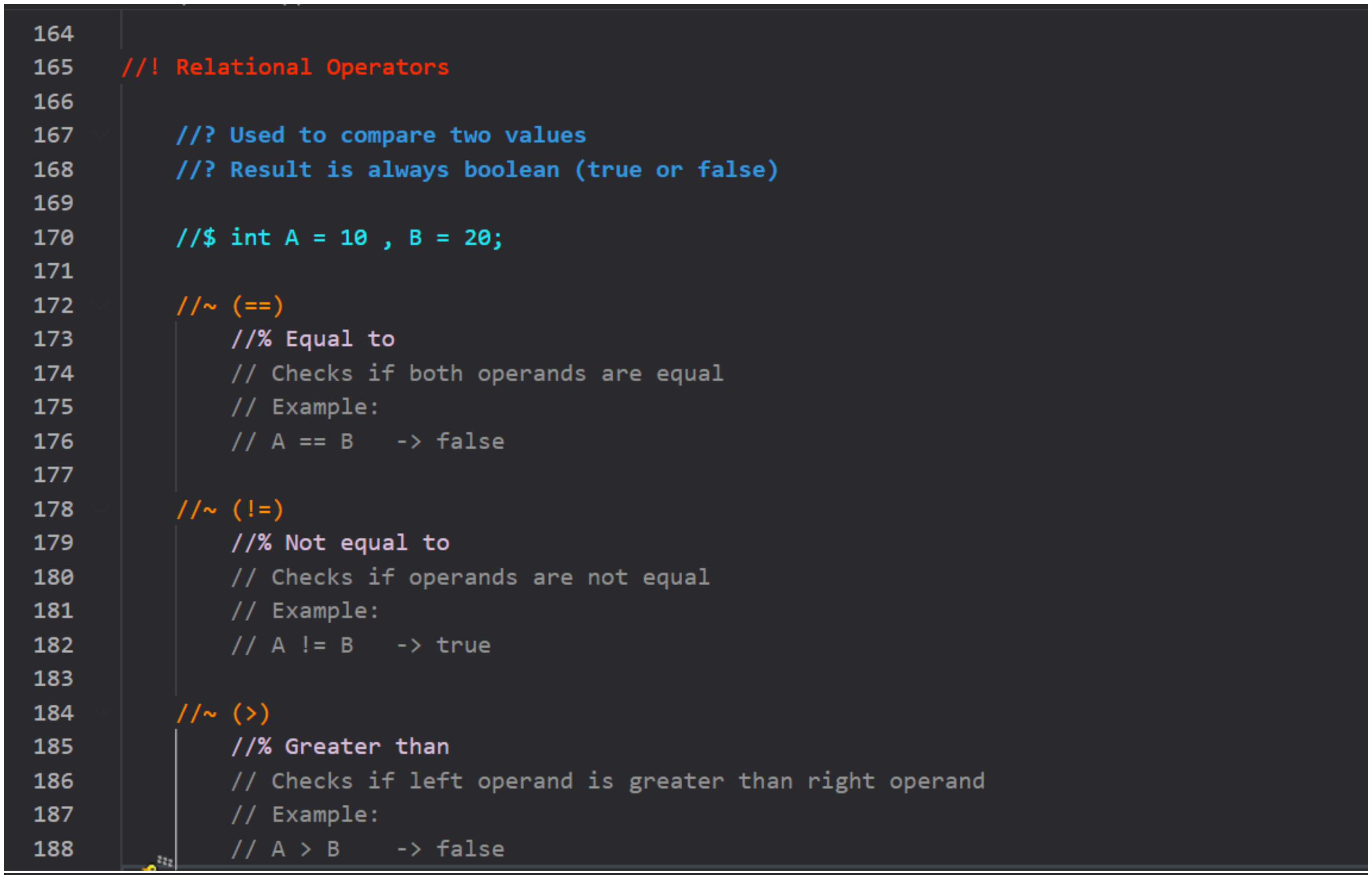
```
    //? Decrement Operator (--)

        //% Pre-decrement (perfix)
            // --x
            // ^ decreases the value first, then uses it
            // Example:
            // x = 5
            // --x -> x becomes 4 immediately

        //% Post-decrement (postfix)
            // x--
            // ^ uses the value first, then decreases it
            // Example:
            // x = 5
            // x-- -> value used is 5, then x becomes 4


//! Important notes about ++ and --

    //? Difference between pre and post operators appears when used inside expressions

        // Example:
        // int x = 5;
        // int y;

        // y = ++x;   // x = 6 , y = 6
        // y = x++;   // y = 5 , x = 6

```



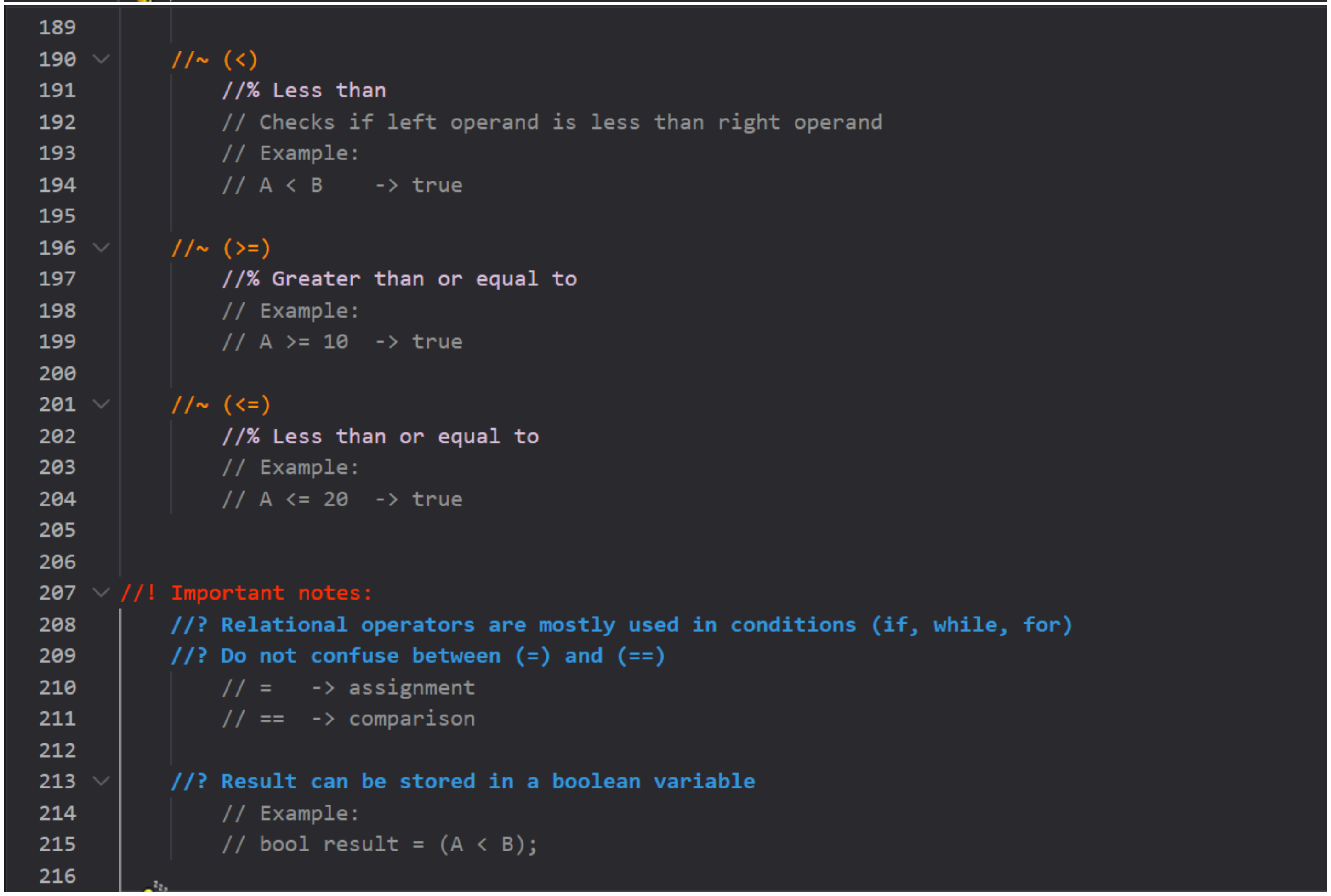
```cpp
        //? Avoid using multiple increments in one expression
            // Example (bad practice):
            // x = x++ + ++x;


    //! Best practice:
        //? Use increment and decrement in simple and clear statements
        //? This improves readability and avoids logical errors



    //! Assignment Operators :

        //? Used to assign values to variables

        //? Basic Assignment Operator (=)
            //% Assigns the value on the right to the variable on the left
            // Example:
            // int x = 10;


        //? Compound Assignment Operators
            //% These operators perform an operation and assignment in one step

            //~ (+=)
                // Adds right operand to left operand and assigns the result
                // Example:
                // x += 5;   // x = x + 5 -> x = 15
```



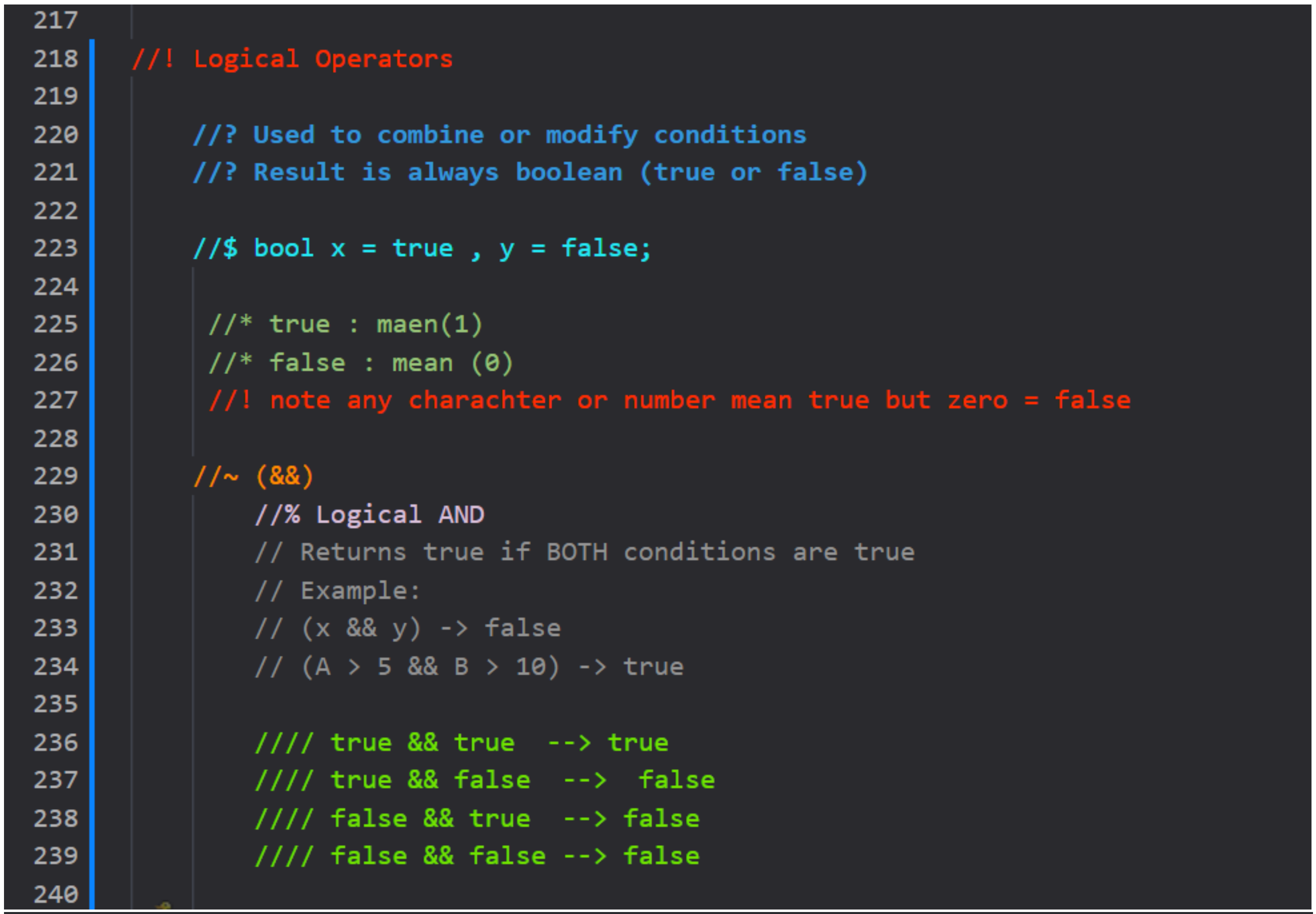
```cpp

            //~ (-=)
                // Subtracts right operand from left operand and assigns the result
                // Example:
                // x -= 3;   // x = x - 3 -> x = 12

            //~ (*=)
                // Multiplies left operand by right operand and assigns the result
                // Example:
                // x *= 2;   // x = x * 2 -> x = 24

            //~ (/=)
                // Divides left operand by right operand and assigns the result
                // Example:
                // x /= 4;   // x = x / 4 -> x = 6

            //~ (%=)
                // Takes remainder using right operand and assigns the result
                // Example:
                // x %= 5;   // x = x % 5 -> x = 1


    //! Important notes:
        //? Compound assignment operators make code shorter and clearer
        //? Result depends on variable type (int, float, double)
        //? Be careful when using /= and %= to avoid division by zero


```



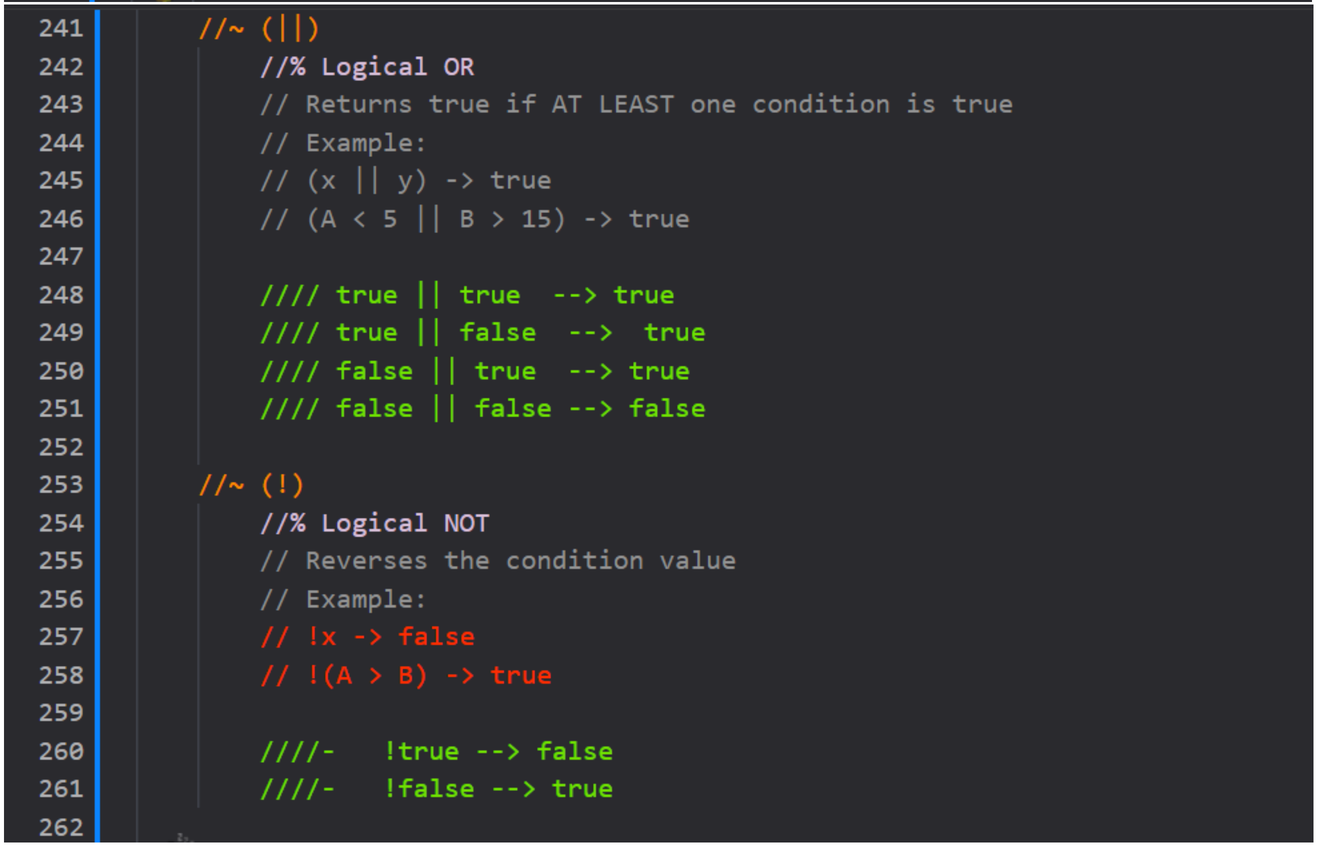
```
//! Relational Operators

    //? Used to compare two values
    //? Result is always boolean (true or false)

    //$ int A = 10 , B = 20;

    //~ (==)
        //% Equal to
        // Checks if both operands are equal
        // Example:
        // A == B   -> false

    //~ (!=)
        //% Not equal to
        // Checks if operands are not equal
        // Example:
        // A != B   -> true

    //~ (>)
        //% Greater than
        // Checks if left operand is greater than right operand
        // Example:
        // A > B    -> false
```



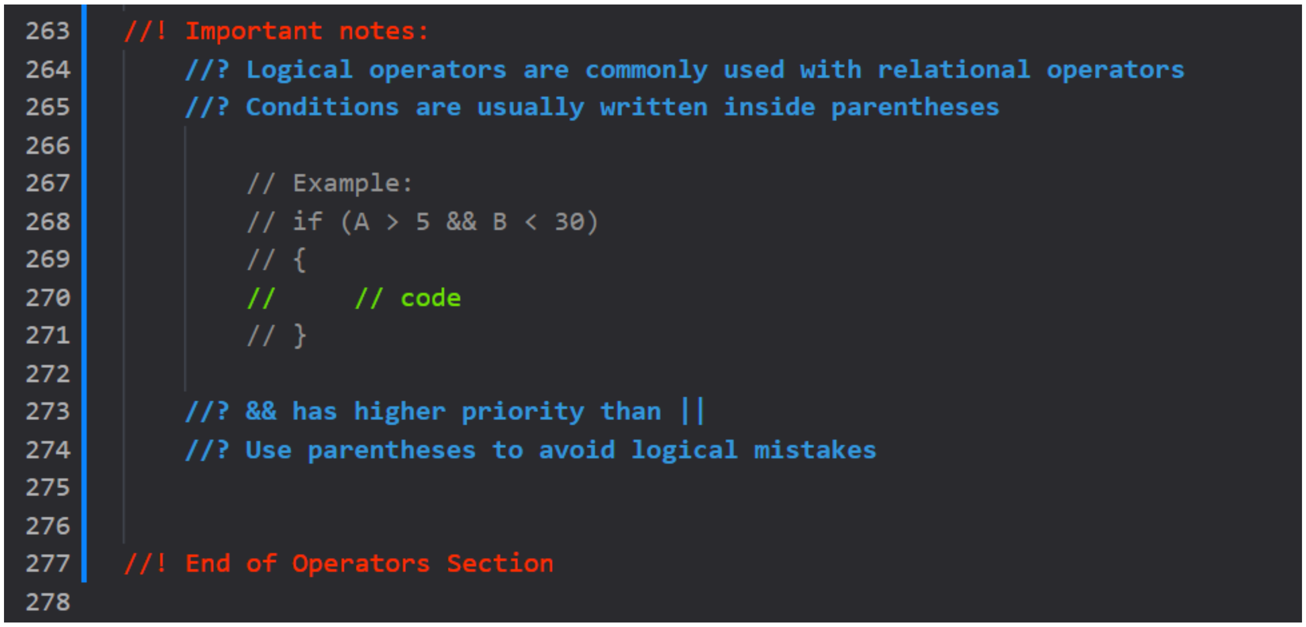
```

    //~ (<)
        //% Less than
        // Checks if left operand is less than right operand
        // Example:
        // A < B    -> true

    //~ (>=)
        //% Greater than or equal to
        // Example:
        // A >= 10  -> true

    //~ (<=)
        //% Less than or equal to
        // Example:
        // A <= 20  -> true


//! Important notes:
    //? Relational operators are mostly used in conditions (if, while, for)
    //? Do not confuse between (=) and (==)
        // =    -> assignment
        // ==   -> comparison

    //? Result can be stored in a boolean variable
        // Example:
        // bool result = (A < B);

```



```

//! Logical Operators

    //? Used to combine or modify conditions
    //? Result is always boolean (true or false)

    //$ bool x = true , y = false;

     //* true : maen(1)
     //* false : mean (0)
     //! note any charachter or number mean true but zero = false

    //~ (&&)
        //% Logical AND
        // Returns true if BOTH conditions are true
        // Example:
        // (x && y) -> false
        // (A > 5 && B > 10) -> true

        //// true && true  --> true
        //// true && false  -->  false
        //// false && true  --> false
        //// false && false --> false

```



```
    //~ (||)
        //% Logical OR
        // Returns true if AT LEAST one condition is true
        // Example:
        // (x || y) -> true
        // (A < 5 || B > 15) -> true

        //// true || true  --> true
        //// true || false  -->  true
        //// false || true  --> true
        //// false || false --> false

    //~ (!)
        //% Logical NOT
        // Reverses the condition value
        // Example:
        // !x -> false
        // !(A > B) -> true

        ////-   !true --> false
        ////-   !false --> true

```



```
//! Important notes:
    //? Logical operators are commonly used with relational operators
    //? Conditions are usually written inside parentheses

        // Example:
        // if (A > 5 && B < 30)
        // {
        //     // code
        // }

    //? && has higher priority than ||
    //? Use parentheses to avoid logical mistakes


//! End of Operators Section

```