

```

You, 1 hour ago | 1 author (You)
1 //! after finishing this file upload it to github
2
3 ∵ //! always search about formalua before start solution
4 //! false mean : 0
5 //! true mean : 1
6
7 #include <iostream>
8 using namespace std;
9
10 ∵ int main()
11 {
12
13 ∵     //! operators:
14         //~ 1) Arithmetic Operators
15         //~ 2) increment & decrement operators
16         //~ 3) Assignment Operators
17         //~ 4) Relational Operators
18
19
20 ∵     //! Arithmetic Operators:
21     //\$ A =10 , B= 20
22         //? (+)
23             //% Adds two operands
24             // A +B = 30
25         //? (-)
26             //% subtracts second operand from the first
27             // A-B = -10

```

```

C++ > C++ 6_Operators.cpp > main()
28     //? (*)
29         //% Multiplies both operands
30         // A*B = 200
31     //? (/)
32         //% divides numerator y de-numerator
33         // B/A = 2
34         // ^ note: if you divide integer by integer the result will be integer
35         // ^ if you want to get float result you have to convert one of them to float
36         // ^ example: (float)A/B = .5
37     //? (%)
38         //% gives remainder of an integer devision
39         //^ A%B = 10
40
41
42
43
44 //! An important note:
45 //? You must put parentheses into the numerator and denominator
46 //? and calculate each term separately, then their result is divided by each other.
47
48 //! note :
49     //// when you want to add 10% to specific number use 10% = 1.1 so
50         // specific number * 1.1
51         //~ 10% = 1.1
52         //~ 16% = 1.16
53         //~ 49 = 1.49
54
55

```

```

56
57 //! increment & decrement operators (++, --)
58
59     ## Used to increase or decrease a variable value by 1
60
61     $$ int x = 5;
62
63     ?? Increment Operator (++)
64
65         // Pre-increment (prefix)
66         // ++x
67         // ^ increases the value first, then uses it
68         // Example:
69         // x = 5
70         // ++x -> x becomes 6 immediately
71
72         // Post-increment (postfix)
73         // x++
74         // ^ uses the value first, then increases it
75         // Example:
76         // x = 5
77         // x++ -> value used is 5, then x becomes 6
78
79
80     ?? Decrement Operator (--)
81
82         // Pre-decrement (prefix)
83         // --x
84         // ^ decreases the value first, then uses it
85         // Example:
86         // x = 5
87         // --x -> x becomes 4 immediately
88
89         // Post-decrement (postfix)
90         // x--
91         // ^ uses the value first, then decreases it
92         // Example:
93         // x = 5
94         // x-- -> value used is 5, then x becomes 4
95
96
97     !!! Important notes about ++ and --
98
99     ?? Difference between pre and post operators appears when used inside expressions
100
101         // Example:
102         // int x = 5;
103         // int y;
104
105         // y = ++x;    // x = 6 , y = 6
106         // y = x++;   // y = 5 , x = 6
107

```

```
108     //? Avoid using multiple increments in one expression
109     // Example (bad practice):
110     // x = x++ + ++x;
111
112
113 //! Best practice:
114 //? Use increment and decrement in simple and clear statements
115 //? This improves readability and avoids logical errors
116
117
118
119 //! Assignment Operators :
120
121     //? Used to assign values to variables
122
123     //? Basic Assignment Operator (=)
124         // Assigns the value on the right to the variable on the left
125         // Example:
126         // int x = 10;
127
128
129     //? Compound Assignment Operators
130         // These operators perform an operation and assignment in one step
131
132         //~ (+=)
133             // Adds right operand to left operand and assigns the result
134             // Example:
135             // x += 5;    // x = x + 5 -> x = 15
136
137     //~ (-=)
138         // Subtracts right operand from left operand and assigns the result
139         // Example:
140         // x -= 3;    // x = x - 3 -> x = 12
141
142     //~ (*=)
143         // Multiplies left operand by right operand and assigns the result
144         // Example:
145         // x *= 2;    // x = x * 2 -> x = 24
146
147     //~ (/=)
148         // Divides left operand by right operand and assigns the result
149         // Example:
150         // x /= 4;    // x = x / 4 -> x = 6
151
152     //~ (%=)
153         // Takes remainder using right operand and assigns the result
154         // Example:
155         // x %= 5;    // x = x % 5 -> x = 1
156
157
158 //! Important notes:
159     //? Compound assignment operators make code shorter and clearer
160     //? Result depends on variable type (int, float, double)
161     //? Be careful when using /= and %= to avoid division by zero
162
163
```

```
164
165 //! Relational Operators
166
167     //? Used to compare two values
168     //? Result is always boolean (true or false)
169
170     //$ int A = 10 , B = 20;
171
172     //~ (==)
173         //% Equal to
174         // Checks if both operands are equal
175         // Example:
176         // A == B -> false
177
178     //~ (!=)
179         //% Not equal to
180         // Checks if operands are not equal
181         // Example:
182         // A != B -> true
183
184     //~ (>)
185         //% Greater than
186         // Checks if left operand is greater than right operand
187         // Example:
188         // A > B -> false
189
190     //~ (<)
191         //% Less than
192         // Checks if left operand is less than right operand
193         // Example:
194         // A < B -> true
195
196     //~ (>=)
197         //% Greater than or equal to
198         // Example:
199         // A >= 10 -> true
200
201     //~ (<=)
202         //% Less than or equal to
203         // Example:
204         // A <= 20 -> true
205
206
207 //! Important notes:
208     //? Relational operators are mostly used in conditions (if, while, for)
209     //? Do not confuse between (=) and (==)
210         // = -> assignment
211         // == -> comparison
212
213     //? Result can be stored in a boolean variable
214         // Example:
215         // bool result = (A < B);
216
```

```
217
218 //! Logical Operators
219
220 //? Used to combine or modify conditions
221 //? Result is always boolean (true or false)
222
223 //\$ bool x = true , y = false;
224
225 /* true : maen(1)
226 /* false : mean (0)
227 //! note any character or number mean true but zero = false
228
229 //~ (&&)
230 //%% Logical AND
231 // Returns true if BOTH conditions are true
232 // Example:
233 // (x && y) -> false
234 // (A > 5 && B > 10) -> true
235
236 ///// true && true --> true
237 ///// true && false --> false
238 ///// false && true --> false
239 ///// false && false --> false
240
241 //~ (||)
242 //%% Logical OR
243 // Returns true if AT LEAST one condition is true
244 // Example:
245 // (x || y) -> true
246 // (A < 5 || B > 15) -> true
247
248 ///// true || true --> true
249 ///// true || false --> true
250 ///// false || true --> true
251 ///// false || false --> false
252
253 //~ (!)
254 //%% Logical NOT
255 // Reverses the condition value
256 // Example:
257 // !x -> false
258 // !(A > B) -> true
259
260 /////- !true --> false
261 /////- !false --> true
262
```

```
263 //! Important notes:  
264 //? Logical operators are commonly used with relational operators  
265 //? Conditions are usually written inside parentheses  
266  
267     // Example:  
268     // if (A > 5 && B < 30)  
269     // {  
270     //     // code  
271     // }  
272  
273 //? && has higher priority than ||  
274 //? Use parentheses to avoid logical mistakes  
275  
276  
277 //! End of Operators Section  
278
```