Contents

Booleans **1**
Truth Tables 1
Precedence and Order of Evaluation 2
Reading and Understanding 2
Computing Simple Boolean Expressions 3
Computing Expressions Involving Booleans and Numerical Values 3

Booleans

This lab serves multiple goals:

- To help you manipulate boolean values,
- To practice boolean operators,
- To understand the concept of *precedence*,
- To practice simple mental calculations.

Truth Tables

1. Copy and paste the following code into the `Main` method of a new project:

```
Console.WriteLine("Conjunction (and, &&) truth table:"
+ "\n\n && \t||  " + true + "\t| " + false
+ "\n--||--|--"
+ "\n" + true + "\t||  " + (true && true) + "\t| " + (true && false)
+ "\n" + false + "\t||  " + (false && true) + "\t| " + (false && false)
+ "\n\n*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*\n");

Console.WriteLine("Negation (not, !) truth table:"
+ "\n\n value \t||  ! "
+ "\n--||-"
+ "\n" + true + "\t||  " + !(true)
+ "\n" + (!true) + "\t||  " + (!false)
+ "\n\n*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*\n");
```

2. Compile and execute it. This should display to the screen the truth tables[1] for conjunction (and, &&) and negation (not, `!`).
3. Make sure you understand both the code and its output.
4. After the truth table for the negation, write code to display the truth tables for three binary operators:

[1] https://www.wikiwand.com/en/Truth_table

(a) the disjunction (or, `||`),
(b) the identity (equality, ==), and
(c) the difference (inequality, `!=`).

Normally, copying the truth table for the conjunction and using the find-and-replace feature of your IDE should make this a quick and easy task.

5. You can make sure you completed this exercise correctly by checking that your output matches the truth tables on Wikipedia for disjunction[2] and equality[3]. To check the inequality truth table, compare your output against the table for exclusive disjunction[4]. Exclusive disjunction (XOR) is conceptually different than inequality but has the same truth table.

Precedence and Order of Evaluation

Reading and Understanding

If you read the documentation on operator precedence[5], you will see that operators are evaluated in a particular order. This order is also given in our notes[6].

For instance, `! true || false && 3 * 2 == 6` will be evaluated as

Operation | | Result | Op. - | - | | – `! true || false && 3 * 2 == 6` | ⇒ | `false || false && 3 * 2 == 6` | `! false || false && 3 * 2 == 6` | ⇒ | `false || false && 6 == 6` | `* false || false && 6 == 6` | ⇒ | `false || false && true` | `== false || false && true` | ⇒ | `false || false` | `&& false || false` | ⇒ | `false` | `||`

Note that an expression like `!3 > 2` does not make any sense: C## would try to take the negation of 3 (since `!` has higher precedence than `>`), but you cannot negate the truth value of an integer! Along the same lines, an expression like `false * true` does not make sense; you can not multiply booleans (what would be "true times false"?)! Similarly, `3 % false` will cause an error; can you see why? These are all examples of "illegal" expressions.

Solution:

`3 % false` would cause an error because the `%` operator (called the

[2] https://www.wikiwand.com/en/Truth_table#Logical_disjunction_(OR)

[3] https://www.wikiwand.com/en/Truth_table#Logical_equality

[4] https://www.wikiwand.com/en/Truth_table#Exclusive_disjunction

[5] https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/#operator-precedence

[6] ../../book.html#precedence-of-operators-1

remainder operator[7]) expects two numerical datatypes, but `false` is not of a numerical datatype, as it is a Boolean.

Computing Simple Boolean Expressions

Evaluate the following expressions. Try to do this "by hand," and write your answers down on paper.

- `true && false || true`
- `!true && false`
- `false || true && !false`
- `false == !true || false`
- `!(true || false || true && true)`
- `!(true || false) && (true && !false)`
- `!true || false && (true && !false)`
- `true != !(false || true)`

Solution:

You can actually use your IDE to check your answers! Simply copy-and-paste the following in a `Main` method:

```
Console.WriteLine("The answers are:\n"
   + "true && false || true: " + (true && false || true) + "\n"
     + "!true && false: " + (!true && false) + "\n"
   + "false || true && !false: " + (false || true && !false) + "\n"
   + "false == !true || false: " + (false == !true || false) + "\n"
   + "!(true || false || true && true): " + (!(true || false || true && true)) + "\n"
   + "!(true || false) && (true && !false): " + (!(true || false) && (true && !false)
   + "!true || false && (true && !false): " + (!true || false && (true && !false)) + "
   + "true != !(false || true): " + (true != !(false || true)) + "\n"
);
```

Computing Expressions Involving Booleans and Numerical Values

For each of the following expressions, decide if it is "legal" or not. If it is, give the result of its evaluation.

- `3 > 2`
- `2 == 4`
- `3 >= 2 != false`
- `3 > false`
- `true && 3 + 5 * 8 == 43`
- `3 + true != false`

Solution:

[7] https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/arithmetic-operators#remainder-operator-

- `3 > 2` is legal (comparing numerical values)
- `2 == 4` is legal (comparing numerical values)
- `3 >= 2 != false` is legal (we first convert `3 >= 2` to `True`, and then test if `true` is different from `false`)
- `3 > false` is *not legal* (a boolean value cannot be less than a numerical value)
- `true && 3 + 5 * 8 == 43` is legal (+ and * are evaluated first, then == compares two numerical values, resulting in a boolean value that can be tested for equality against `true`)
- `3 + true != false` is *not legal* (+ is evaluated first, but a numerical value and a boolean cannot be summed).