CSCI 1301 - Lab 13

Clément Aubert

March 22, 2018

Deadlines: Part I and Part II are required to be able to turn in Project #3, presented in Part III.

Part I – Truth Tables

Copy-and-paste the following code in the `Main` method of a new project:

```
/*
 * We have two boolean values: true and false.
 * We can use the constant "true" and "false",
 * we can also declare constants with the same value,
 * but a shorter name:
 */
const bool t = true;
const bool f = false;

Console.WriteLine("Conjonction (and, &&) truth table:"
+ "\n\n\t" + t+ "\t" + f
+ "\n" + t+ "\t" + (t && t)+ "\t" + (t && f)
+ "\n" + f+ "\t" + (f && t)+ "\t" + (f && f)
+ "\n\n*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*\n");

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

Compile and execute it.

Then, write the code that will display on the screen the truth tables for the binary operators disjunction (or, `||`), identity (equality, `==`) and difference (inequality, `!=`).

Normally, using the find-and-replace feature of VS should make this task easy and fast.

Part II – Precedence and Order of Evaluation

Reading and Understanding

If you look at https://docs.microsoft.com/en-us/cpp/c-language/precedence-and-order-of-evaluation, you will see that

`!`	is evaluated before
`*`, `/`, and `%`	which are evaluated before
`+` and `-`	which are evaluated before
`<`, `>`, `<=`, and `>=`	which are evaluated before
`==` and `!=`	which are evaluated before
`&&`	which is evaluated before
`||`	which comes last

and that within those groups, operations are evaluated from left to right.

So that, for instance, `! true || false && 3 * 2 == 5` will be evaluated as

`! 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` doesn't make any sense: C# would try to take the negation of `3`, but you can't negate an integer! Along the same way, an expression like `false * true` doesn't make any sense: you can't multiply booleans! Similarly, `3 % false` will cause an error: can you decide why?

Computing Simple Boolean Expressions

Evaluate the following expressions (where `t` stands for `true`, and `f` for `false`):

- `t && f || t`
- `!t && f`
- `f || t && !f`
- `f == !t || f`
- `!(t || f || t && t)`
- `!(t || f) && (t && !f)`
- `!t || f && (t && !f)`
- `t != !(f || t)`

Computing Expressions Involving Booleans and Numerical Values

For each of the following expression, decide if they are "legal" or not. If they are, give the result of their evaluation.

- `3 > 2`
- `2 == 4`
- `3 >= 2 != f`
- `3 > f`
- `t && 3 + 5 * 8 == 43`
- `3 + t != f`

Part III – A Guessing Game

Write a program that

a. Store your favorite number in a variable
b. Ask the user to enter a numerical value, and store the user's answer in a variable.
c. With an `if` statement, display on the screen "You guessed correctly" if number entered by the user is your favorite number.
d. Once this part of the program works, add an `if` statement that displays on the screen "Try a lower value!" if the number entered by the user is strictly greater than your favorite number.
e. Once this part of the program works, add an `if` statement that displays on the screen "Try a greater value!" if the number entered by the user is strictly lower than your favorite number.
f. Once this part of the program works, add an `if` statement that displays on the screen "You found a multiple of my favorite number!" if the number entered by the user is a multiple of your favorite number, but different from it.

Part IV – Pushing Further (Optional)

This lab's "Pushing Further" combines Part 1 and our previous study of static class members. The goal is to design a **static class** that will be used to return truth tables as string.

a. Start by reading about static classes at https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/static-classes-and-static-class-members
b. Ask yourself: why would we want the class that contains methods to return truth tables to be static?
c. Start writing the class: start with `static class TruthTable`, add two private constants for the two boolean values. No need to declare them static: a `const` attribute of a class is always static!
d. Write the first method: a static method that returns a string, the truth table for the conjonction, takes no argument, and is called `And`.
e. Make sure your class is working: in the `Main` method, call your method, using `TruthTable.And()`, and display on the screen the string it returned.
f. Write the methods that returns the other truth tables.

g. Write a static method that takes a string argument, and returns the truth table for the conjonction if the argument is "and" or "&&", the disjunction if the argument is "or" or "||", etc. You can use `if`, `if-else`, or `switch`, to do so. Have a look at the documentation (for instance, https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/if-else for `if-else`, https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/switch for switch).