Lesson 19 - Logical Operators

Recall,

> ### With Guard Clause
>
> ```
> int age = 20;
> double height = 6;
> double weight = 65;
>
> if (age < 18) return;
> if (age > 30) return;
> if (height < 5.5) return;
> if (weight < 60) return;
> if (weight > 80) return;
>
> Console.WriteLine("You pass");
> ```

In the above guard clause, we see the problem that even though we have multiple conditions, the code executed based on each condition is the same, so in situations like these, it is better to combine conditions to make the code more concise and clear.

Logical Operators

Logical operators are used to join multiple conditions and return `true` or `false` combined result for the combined conditions. There are 3 main logical operators.

Operators	Name of Operator	Form
`&&`	**AND** operator	`x && y`
`||`	**OR** operator	`x || y`
`!`	**NOT** operator	`!x`

For the sake of demonstration, lets assume that `true = 1` and `false = 0`

AND table

x	y	x && y
0	0	0
0	1	0
1	0	0
1	1	1

AND requires that all sub conditions be `true`, for the result to be `true`. If any of the sub conditions is `false`, then the final result will be `false`. When would we want to use this? In a situation, lets say we can only let people of age greater than 18 years AND height greater than 5.5 feet on a ride in an amusement park, then we could use this condition. Lets take an example:

```
void CheckEligibility(int age, double height)
{
        if (age >= 18 && height >= 5.5)
                Console.WriteLine("You are eligible.");
        else
                Console.WriteLine("You are not eligible.");
}

CheckEligibility(20, 6);
CheckEligibility(20, 5);
```

```
OUTPUT:
You are eligible.
You are not eligible.
```

You can also join more than two conditions. Let us suppose that people of age 60 years and above cannot get on our ride due to risk of shock, then we have 3 conditions.

```
void CheckEligibility(int age, double height)
{
        if (age >= 18 && age < 60 && height >= 5.5)
                Console.WriteLine("You are eligible.");
        else
                Console.WriteLine("You are not eligible.");
}
```

```
CheckEligibility(68, 6.2);
CheckEligibility(20, 6);
```

```
OUTPUT:
You are not eligible.
You are eligible.
```

OR table

x	y	x || y
0	0	0
0	1	1
1	0	1
1	1	1

OR will result in a final `true` answer if any of the sub conditions are true. Why would we want to use this? If we can let people of age 18 and above on the ride OR children under 18 if they come with an adult.

```
void CheckEligibility(int age, bool withAdult)
{
  if (age >= 18 || withAdult)
        Console.WriteLine("You are eligible.");
  else
        Console.WriteLine("You are not eligible.");
}

CheckEligibility(20, false);
CheckEligibility(20, true);
CheckEligibility(12, true);
CheckEligibility(12, false);
```

```
OUTPUT:
You are eligible.
You are eligible.
You are eligible.
You are not eligible.
```

Similarly, you can join multiple **OR** operators as well.

```
void CheckEligibility(int age, bool withAdult, bool isEmployee)
{
  if (age >= 18 || withAdult || isEmployee)
        Console.WriteLine("You are eligible.");
  else
        Console.WriteLine("You are not eligible.");
}

CheckEligibility(20, false);
CheckEligibility(20, true);
CheckEligibility(12, true);
CheckEligibility(12, false);
```

You can also join multiple AND and OR operators with each other. For example, we can allow a person if they are 18 or over, but not if they are 60 and above, and if they are under 18, they can come if accompanied by an adult.

```
void CheckEligibility(int age, bool withAdult)
{
  if ((age >= 18 || withAdult) && age < 60)
          Console.WriteLine("You are eligible.");
  else
          Console.WriteLine("You are not eligible.");
}

CheckEligibility(20, false);  // over 18

CheckEligibility(12, false);  // under 18
CheckEligibility(12, true);   // under 18 with adult

CheckEligibility(65, false);  // over 60
CheckEligibility(65, true);   // over 60 with adult
```

```
OUTPUT:
You are eligible.
You are not eligible.
You are eligible.
You are not eligible.
You are not eligible.
```

The **NOT** operator is a unary operator, so it works on a single value. It reverses the logical state of any value.

NOT table

x	!x
0	1
1	0

Sometimes it is easier to just reverse a condition, or you may have boolean values on which the result depends on, but on the false condition or reverse of a multipart condition.

```
void CheckEligibility(int age)
{
  if ( !(age < 18 || age >= 60) )
          Console.WriteLine("You are eligible.");
  else
          Console.WriteLine("You are not eligible.");
}

CheckEligibility(20);  // over 18
CheckEligibility(12);  // under 18
CheckEligibility(65);  // over 60
```

```
void CheckEligibility(int age)
{
        bool isMinor = false;
        if (age < 18)
                isMinor = true;

        while (isMinor)
        {
                Console.WriteLine(age++);
                if (age >= 18)
                        isMinor = !isMinor;
        }
}

CheckEligibility(5);
```

Now converting the guard clause:

```
void GuardClause(int age, double height, double weight)
{
        if ( !(age >= 18 && age <= 30) ||
```

```
                        height < 5.5 ||
                        !(weight >= 60 && weight <= 80) )
                return;
}

GuardClause(18, 6, 74);
```