C# Enumerable.Range in LINQ

LINQ's Enumerable.Range to generate a sequence of consecutive numbers

When you need to generate a sequence of numbers in ascending order, you can just use a `while` loop with an enumerator, or you can use `Enumerable.Range`.

This method, which you can find in the `System.Linq` namespace, allows you to generate a sequence of numbers by passing two parameters: the *start* number and the *total numbers* to add.

```
Enumerable.Range(start:10, count:4) // [10, 11, 12, 13]
```

⚠ Notice that the second parameter is **not** the last number of the sequence. Rather, it's the length of the returned collection.

Clearly, it also works if the `start` parameter is negative:

```
Enumerable.Range(start:-6, count:3) // [-6, -5, -4]
```

But it will not work if the `count` parameter is negative: in fact, it will throw an `ArgumentOutOfRangeException`:

```
Enumerable.Range(start:1, count:-23) // Throws ArgumentOutOfRangeException
// with message "Specified argument was out of the range of valid values"(Parameter 'count')
```

⚠ Beware of overflows: it's not a circular array, so if you pass the `int.MaxValue` value while building the collection you will get another `ArgumentOutOfRangeException`.

```
Enumerable.Range(start:Int32.MaxValue, count:2) // Throws ArgumentOutOfRangeException
```

💡 Smart tip: you can use `Enumerable.Range` to generate collections of other types! Just use LINQ's `Select` method in conjunction with `Enumerable.Range`:

```
Enumerable.Range(start:0, count:5)
    .Select(_ => "hey!"); // ["hey!", "hey!", "hey!", "hey!", "hey!"]
```

Notice that this pattern is not very efficient: you first have to build a collection with N integers to then generate a collection of N strings. If you care about performance, go with a simple `while` loop - if you need a quick and dirty solution, this other approach works just fine.