Delegates exercises

1. Define delegate type named **StringJoin** with two string parameters and return-type string.
   a. Verify by creating a variable of type StringJoin and set it to reference a method such as:

   ```
   string ConcatString(string l, string r) { return l + r; }
   ```

   Verify that invoking the delegatevariable with the parameters "hello " "delegates" results in the string "hello delegates"

2. Define a method JoinThree parameterized with three strings and a StringJoin, which joins the strings from left to right. For example:

   ```
   a. Join3("a", "b", "c", (l, r) => l + r); // abc
   b. Join3("a", "b", "c", (l, r) => l + "." + r); // a.b.c
   c. Join3("a", "b", "c", (l, r) => l); // a
   ```

3. Define a method JoinAllStrings, parameterized by a list of strings and a StringJoin, which joins all strings in the list from left to right. For example:

```
JoinAllStrings(new List<string>{"a","b","c","d"},(l, r) => l + "." + r) // a.b.c.d
JoinAllStrings(new List<string>{"a","b","c","d"},(l, r) => l + r) // abcd
JoinAllStrings(new List<string>{"a","b","c","d"},(l, r) => r) // d
```

4. Define a generic delgate type named **Join** with a single type parameter T, with two parameters and return-type of type T.
   d. Use the Join delegate to implement a generic method JoinList which as the JoinAll method, joins all elements in the list, as defined by the Join parameter. Examples:

   ```
   JoinAll(new List<int> { 1, 2, 3, 4 }, (a, b) => a + b) // 10
   JoinAll(new List<int> { 1, 2, 3, 4 }, (a, b) => a * b) // 24
   JoinAll(new List<string>{"a","b","c","d"},(l, r) => l + "." + r) // a.b.c.d
   ```

5. Write a generic method Exists(Predicate<T> f, T[] a) that takes a type parameter T and two arguments: a unary lambda expression f and an array a of type T. The method should return true if the array contains an element for which the predicate evaluates to true. Otherwise, it should return false.
6. Write a generic method twice(**DELEGATETYPE** f, T v) with type parameter T and two arguments: a DELEGATETYPE from the standard library f and a value v of type T. The method should return the result of applying f twice to the argument. For example, twice(x -> x * 2, 1) = 4. Choose the appropriate type as DELEGATETYPE. See hint on last page.[i]
7. EXTRA: Investigate and try out LINQ as an extra exercise – start with **Where**, **Select,** and **GroupBy** and combine these! – to begin with, use Method syntax, not query syntax.

---

[i] DELEGATETYPE could be of type Func<> (why not Action?). You have to decide parameters of Func to match the lambda expression in the example.