Операции Take, TakeWhile, Skip и SkipWhile

Операции разбиения (partitioning) позволяют вернуть выходную последовательность, которая является подмножеством входной последовательности.

Take

Операция Take возвращает указанное количество элементов из входной последовательности, начиная с ее начала. Операция Take имеет один прототип, описанный ниже:

C#

```
public static IEnumerable<T> Take<T>(
    this IEnumerable<T> source,
    int count);
```

Этот прототип указывает, что Take принимает входную последовательность и целое число count, задающее количество элементов, которые нужно вернуть, и возвращает объект, который при перечислении выдает первые count элементов из входной последовательности.

Если значение count больше количества элементов во входной последовательности, тогда каждый элемент из нее попадает в выходную последовательность.

Ниже показан пример использования операции Take:

Пройди тесты

C#

C# тест (легкий) (https://professorweb.ru/test/c-sharp-test.html)

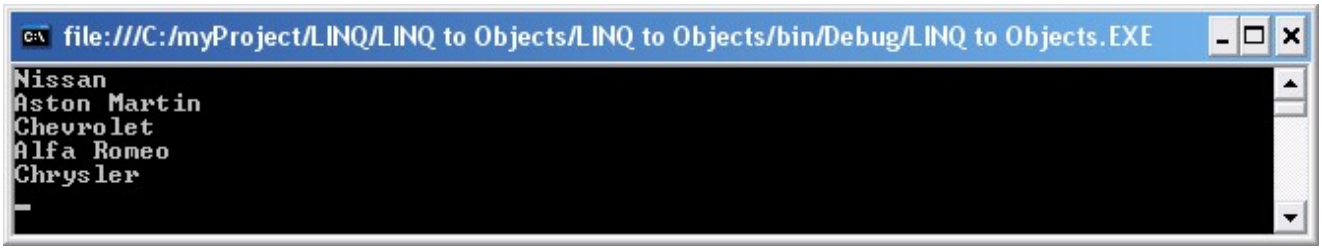
.NET тест (средний) (https://professorweb.ru/test/asp-test.html)

```
string[] cars = { "Nissan", "Aston Martin", "Chevrolet", "Alfa Romeo", "Chrysler", "Dodge",
"BMW",
        "Ferrari", "Audi", "Bentley", "Ford", "Lexus", "Mercedes", "Toyota", "Volvo", "Suba
ru", "Жигули :)"};

IEnumerable<string> auto = cars.Take(5);

foreach (string str in auto)
        Console.WriteLine(str);
```

Этот код вернет первые пять элементов из массива cars. Полученный результат выглядит следующим образом:



TakeWhile

Операция TakeWhile возвращает элементы из входной последовательности, пока истинно некоторое условие, начиная с начала последовательности. Остальные входные элементы пропускаются.

Операция Take While имеет два прототипа, описанные ниже:

Первый прототип TakeWhile

C#

```
public static IEnumerable<T> TakeWhile<T>(
       this IEnumerable<T> source,
       Func<T, bool> predicate);
```

Эта операция TakeWhile принимает входную последовательность и делегат метода-предиката, а возвращает объект, перечисление по которому выдает элементы до тех пор, пока метод-предикат не вернет false. Метод-предикат принимает элементы по одному из входной последовательности и возвращает признак того, должен элемент включаться в выходную последовательность или нет. Если да, обработка входных элементов продолжается. Как только метод-предикат вернет false, никакие последующие входные элементы не обрабатываются.

Второй прототип TakeWhile

C#

```
public static IEnumerable<T> TakeWhile<T> (
       this IEnumerable<T> source,
       Func<T, int, bool> predicate);
```

Этот прототип подобен первому, за исключением того, что метод-предикат получает вдобавок индекс элемента из входной последовательности, начинающийся с нуля.

Ниже приведен пример вызова первого прототипа:

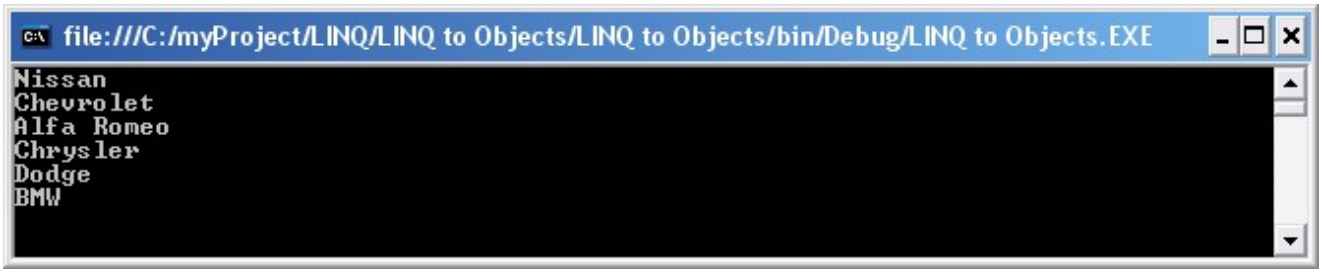
C#

```
string[] cars = { "Nissan", "Chevrolet", "Alfa Romeo", "Chrysler", "Dodge", "BMW", "Aston M
artin",
                            "Ferrari", "Audi", "Bentley", "Ford", "Lexus", "Mercedes", "Toy
ota", "Volvo", "Subaru", "Жигули :)"};

IEnumerable<string> auto = cars.TakeWhile(s => s.Length < 12);

foreach (string str in auto)
        Console.WriteLine(str);
```

В приведенном коде входные элементы извлекаются до тех пор, пока их длина не превышает 11 символов. Ниже показан результат:



Элемент, который заставил операцию TakeWhile прекратить обработку входной последовательности — Aston Martin. Рассмотрим пример второго прототипа операции TakeWhile:

Пройди тесты

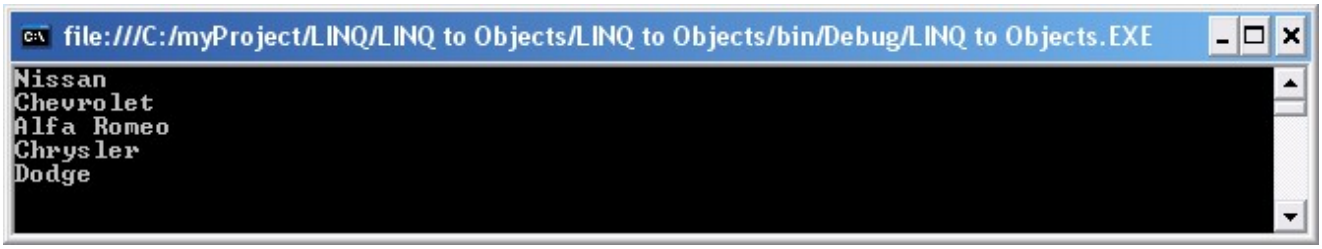
C# C# тест (легкий) (https://professorweb.ru/test/c-sharp-test.html)

.NET тест (средний) (https://professorweb.ru/test/asp-test.html)

```
IEnumerable<string> auto = cars.TakeWhile((s, i) => s.Length < 12 && i < 5);

foreach (string str in auto)
        Console.WriteLine(str);
```

Код в этом примере прекращает выполнение, когда входной элемент превысит 11 символов в длину или когда будет достигнут шестой элемент — в зависимости от того, что произойдет раньше. Вот результат:



В этом случае обработка остановилась по достижении шестого элемента.

Skip

Операция Skip пропускает указанное количество элементов из входной последовательности, начиная с ее начала, и выводит остальные.

Операция Skip имеет один прототип, описанный ниже:

C#

```
public static IEnumerable<T> Skip<T>(
          this IEnumerable<T> source, int count);
```

Операция Skip получает входную последовательность и целое число count, задающее количество входных элементов, которое должно быть пропущено, и возвращает объект, который при перечислении пропускает первые count элементов и выводит все последующие элементы.

Ниже приведен пример вызова операции Skip:

Пройди тесты

C#

C# тест (легкий) (https://professorweb.ru/test/c-sharp-test.html)

.NET тест (средний) (https://professorweb.ru/test/asp-test.html)

```
string[] cars = { "Nissan", "Chevrolet", "Alfa Romeo", "Chrysler", "Dodge", "BMW", "Aston M
artin",
                            "Ferrari", "Audi", "Bentley", "Ford", "Lexus", "Mercedes", "Toy
ota", "Volvo", "Subaru", "Жигули :)"};

IEnumerable<string> auto = cars.Skip(5);

foreach (string str in auto)
        Console.WriteLine(str);
```

В данном примере пропускаются первые 5 элементов. Обратите внимание, что в следующем выводе действительно пропущены первые пять элементов входной последовательности:

SkipWhile

Операция SkipWhile обрабатывает входную последовательность, пропуская элементы до тех пор, пока условие истинно, а затем выводит остальные в выходную последовательность. У операции SkipWhile есть два прототипа, описанные ниже:

C#

```
public static IEnumerable<T> SkipWhile<T>(
      this IEnumerable<T> source,
      Func<T, bool> predicate);
```

Операция SkipWhile принимает входную последовательность и делегат метода-предиката, а возвращает объект, который при перечислении пропускает элементы до тех пор, пока метод-предикат возвращает true. Как только метод-предикат вернет false, операция SkipWhile начинает вывод всех прочих элементов. Метод-предикат принимает элементы входной последовательности по одному и возвращает признак того, должен ли элемент быть выведен в выходной последовательности.

Второй прототип SkipWhile

C#

```
public static IEnumerable<T> SkipWhile<T>(
     this IEnumerable<T> source,
     Func<T, int, bool> predicate);
```

Этот прототип подобен первому во всем, за исключением дополнительного параметра — индекса элемента из входной последовательности, начинающегося с нуля.

Ниже приведен пример вызова первого прототипа операции SkipWhile:

C#

```
string[] cars = { "Alfa Romeo", "Aston Martin", "Audi", "Nissan", "Chevrolet",  "Chrysler",
"Dodge", "BMW",
                            "Ferrari",  "Bentley", "Ford", "Lexus", "Mercedes", "Toyota",
"Volvo", "Subaru", "Жигули :)"};

IEnumerable<string> auto = cars.SkipWhile(s => s.StartsWith("A"));

foreach (string str in auto)
        Console.WriteLine(str);
```

В этом примере метод SkipWhile должен пропускать элементы до тех пор, пока они начинаются с буквы "A". Все остальные элементы выдаются в выходную последовательность. Результат предыдущего запроса выглядит так:

Теперь рассмотрим пример использования второго прототипа SkipWhile:

C#

```
string[] cars = { "Alfa Romeo", "Aston Martin", "Audi", "Nissan", "Chevrolet",  "Chrysler",
"Dodge", "BMW",
                            "Ferrari",  "Bentley", "Ford", "Lexus", "Mercedes", "Toyota",
"Volvo", "Subaru", "Жигули :)"};

IEnumerable<string> auto = cars.SkipWhile((s, i) => s.StartsWith("A") && i < 10);

foreach (string str in auto)
        Console.WriteLine(str);
```

В данном примере входные элементы пропускаются до тех пор, пока они начинаются с буквы "A" или пока не будет достигнут десятый элемент. Остальные элементы выдаются в выходную последовательность.

Пропуск элементов был прекращен, как только встретился элемент Nissan, поскольку он начинается с N, хотя его индексом является 3.

Назад (2_2.php)	2	3	4	Вперед (2_4.php)

Пройди тесты

C# тест (легкий) (https://professorweb.ru/test/c-sharp-test.html)

.NET тест (средний) (https://professorweb.ru/test/asp-test.html)