Немного инфы о том, что такое список и зачем он нужен.

Помимо массивов в языке C# существует такая структура данных как список. Хотя списки и работают медленнее массивов (к примеру, получить элемент массива по индексу можно за $O(1)$, тогда как в списке только за $O(n)$, где n – длина списка), их использование зачастую может значительно облегчить вам жизнь.

Списки реализованы классом **`List<T>`**, где **`T`** – тип объектов, которые необходимо закинуть в список. Этот класс является обобщенным. Тема обобщений будет затронута в дальнейшем курсе, но стоит отметить несколько их преимуществ:

- Обобщения избавляют от необходимости использования приведения типов и ручной проверки соответствия типов в коде программы. Поэтому **не стоит** создавать список типа **`Object`**, поскольку это не обеспечит безопасность выполнения всех операций при дальнейшей работе с его объектами.
- Обобщения позволяют писать один код и применять его к обширному количеству типов.

Чтобы использовать списки, необходимо подключить библиотеку **`System.Collections.Generic`**.

Список можно объявить следующим образом:

- `List<int> list = new List<int>(); // Пустой список, вместимость которого равна 0`
- `List<double> list = new List<double>(10); // Пустой список вместимости (Capacity) 10`
- ```
  Int32[] array = new[] { 1, 1, 2, 3, 5 };
  // Список на основе массива из 5 элементов вместимости 5
  List<int> list = new List<int>(array);
  ```

У класса **`List<T>`** есть два основных свойства: **`Capacity`** и **`Count`**. Крайне **важно** осознавать, что это не одно и то же и как они вообще различаются между собой.

- **`Capacity`** устанавливает или получает общее количество элементов, которое внутренняя структура данных списка может вместить без изменения объема (условно, вместимость/емкость списка)
- **`Count`** получает количество элементов, которые содержатся в списке.

Очень часто (почти всегда) при попытке использовать **`Capacity`** вместо **`Count`** программа выкидывает необработанное исключение **`ArgumentOutOfRangeException`**. К примеру, это произойдет, если написать вот такой код:

```
using System;
using System.Collections.Generic;

    class Program
    {
        static Random random = new Random();
        static void Main(string[] args)
        {
            List<int> list = new List<int>(15);
            for (int i = 0; i < 10; i++)
                list.Add(random.Next(0, 15));
             // Необработанное исключение ArgumentOutOfRangeException
            for (int i = 0; i < list.Capacity; i++)
                Console.Write(list[i] + "\t");
        }
    }
```

NOTE: Для корректной работы этой программы в цикле **`for`** следовало бы использовать **`list.Count`**, либо применить **`foreach`** для перечисления элементов списка.
```

Если количество элементов в списке равно **`Capacity`**, добавление элемента в список приведет к увеличению его емкости в два раза.

Важное замечание: не нужно пытаться очистить список путем обнуления значения **`Capacity`** (спойлер: это плохо закончится). Лучше вообще **не трогать** свойство **`Capacity`** и пользоваться только **`Count`**, тем более что вместимость списка автоматически увеличивается при добавлении в него элементов.

Следующий код при попытке прохождения по списку вызовет **`ArgumentOutOfRangeException`**:

```
using System;
using System.Collections.Generic;

class Program
{
    static Random random = new Random();
    static void Main(string[] args)
    {
        List<int> list = new List<int>();
        for (int i = 0; i < 8; i++)
            list.Add(random.Next(0, 15));
        list.Capacity = 0;
        // Необработанное исключение ArgumentOutOfRangeException: емкость меньше текущего
размера
        foreach (var item in list)
            Console.Write(item + "\t");
    }
}
```

Если возникает необходимость очистки списка следует использовать метод **`Clear()`**, удаляющий все элементы списка, обнуляющий значение **`Count`** и не изменяющий значение **`Capacity`**.

Довольно приятный факт: удалять элемент из списка можно с любой позиции, причем для этого не нужно «танцевать с бубном», как это происходит при использовании массивов. При удалении элемента из списка, индексы последующих элементы сдвигаются на единицу автоматически.

Вообще у класса **`List<T>`** для добавления и удаления элементов из списка существуют такие методы как:

- **`Add(T item)`** – добавляет элемент **`item`** в конец списка
- **`AddRange(IEnumerable<T> collection)`** – добавляет перечислимую коллекцию (например, список или массив) в конец списка
- **`Remove(T item)`** – удаляет **первое** вхождение элемента **`item`** в список
- **`RemoveAt(Int32 index)`** – удаляет элемент с заданным индексом из списка
  Аларм: кидает **`ArgumentOutOfRangeException`**, если индекс меньше нуля, либо больше количества элементов в списке (что довольно очевидно, но следует помнить)
- **`Insert(Int32 index, T item)`** – вставляет элемент **`item`** на позицию **`index`**

Для получения сведений о наличии того или иного элемента в списке есть:

- **`Contains(T item)`** – определяет, содержится ли элемент **`item`** в списке
- **`IndexOf(T item)`** – ищет индекс **первого** вхождения элемента **`item`** в список (если такой элемент отсутствует вернет -1)
- **`LastIndexOf(T item)`** – ищет индекс **последнего** вхождения элемента **`item`** в список

…а также некоторое количество перегрузок этих методов, которые можно откопать на MSDN.

Еще можно отметить встроенные методы для реверсирования порядка элементов списка **`Reverse()`** и сортировки по возрастанию **`Sort()`**.

Когда удобно использовать списки?

- При необходимости вставки элементов куда-либо, кроме конца последовательности объектов (на самом деле даже при добавлении элементов в конец списком пользоваться удобнее, чем постоянно вызывать метод **`Resize`** для массива)
- При необходимости удаления элементов из последовательности с любой позиции
- Для быстрого определения вхождения заданных элементов в последовательность

Примерчик:

```
using System;
using System.Collections.Generic;

/* Сгенерировать последовательность случайной длины, состоящую из произвольных целых чисел.
 * Удалить из последовательности все элементы кратные 3. */
class Program
{
    static Random random = new Random();
    static void Main(string[] args)
    {
        int n = random.Next(5, 10);
        List<int> sequence = new List<int>();
        Console.WriteLine("Исходная последовательность: ");
        for (int i = 0; i < n; i++)
        {
            int randomNumber = random.Next(15, 80);
            sequence.Add(randomNumber);
            Console.Write(randomNumber + "\t");
        }

        for (int i = 0; i < sequence.Count; i++)
            if (sequence[i] % 3 == 0)
            {
                sequence.RemoveAt(i);
                i--;
            }
        Console.WriteLine("\nИзмененная последовательность: ");
        foreach (var number in sequence)
            Console.Write(number + "\t");
    }
}
```