Лабораторная работа №4

«Массивы, интерфейсы IComparer<T> и IEnumerator<T>, оператор yield»

**Цели работы:**

1. Научиться синтаксису и принципам работы с массивами средствами языка C#.
2. Научиться реализовывать интерфейсы IComparer<T> и IEnumerator<T>.
3. Получить практические навыки работы с оператором yield.

**Задание№1**

Создайте класс MyMatrix, представляющий матрицу m на n.

Создайте конструктор, принимающий число строк и столбцов, заполняющий матрицу случайными числами в диапазоне, который пользователь вводит при запуске программы.

Определите операторы сложения, вычитания и умножения матриц, а также умножения и деления матрицы на число.

Создайте пользовательский индексатор матрицы для доступа к элементам матрицы по номеру строки и столбца.

**Задание№2**

Создайте класс Car с тремя авто-свойствами: Name, ProductionYear и MaxSpeed, соответствующими названию, году выпуска и максимальной скорости соответственно.

Создайте класс CarComparer : IComparer<Car> и реализуйте метод Compare таким образом, чтобы можно было сортировать массив элементов Car по названию, году выпуска или максимальной скорости по выбору.

Создайте массив элементов Car и продемонстрируйте сортировку различными способами.

**Задание№3**

Используйте класс Car из задания №2, на его основе создайте класс CarCatalor, содержащий массив элементов типа Car.

Для класса CarCatalog реализуйте возможность итерации по элементам массива Car с помощью оператора foreach различными способами:

1. Прямой проход с первого элемента до последнего.
2. Обратный проход от последнего к первому.
3. Проход по элементам массива с фильтром по году выпуска.
4. Проход по элементам массива с фильтром по максимальной скорости.

Примечание: для выполнения задания необходимо реализовать различные итераторы, используя конструкцию yield return. Для п.3 и 4, итератор должен принимать год выпуска и скорость как параметр, чтобы возвращать только те элементы коллекции, которые удовлетворяют условию.

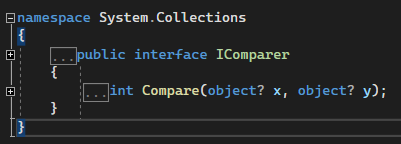
**Теоретические сведения**

**Реализация интерфейса IComparer**

Для сортировки объектов определяемых пользователем классов зачастую проще всего реализовать в этих классах интерфейс IComparable. Тем не менее данную задачу можно решить и с помощью интерфейса IComparer. Для этой цели необходимо сначала создать класс, реализующий интерфейс IComparer, а затем указать объект этого класса, когда потребуется сравнение.

Интерфейс IComparer существует в двух формах: обобщенной(generic) и необобщенной. Несмотря на сходство применения обеих форм данного интерфейса, между ними имеются некоторые, хотя и небольшие, отличия.

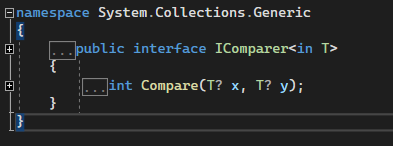
В необобщенном интерфейсе IComparer определяется только один метод Compare():



В методе Compare() сравниваются объекты х и у. Для сортировки объектов по нарастающей конкретная реализация данного метода должна возвращать нулевое значение, если значения сравниваемых объектов равны; положительное — если значение объекта х больше, чем у объекта у; и отрицательное — если значение объекта х меньше, чем у объекта у. А для сортировки по убывающей можно обратить результат сравнения объектов. Если же тип объекта x не подходит для сравнения с объектом y, то в методе Compare() может быть сгенерировано исключение ArgumentException.

Объект типа IComparer может быть указан при конструировании объекта класса SortedList, при вызове метода ArrayList.Sort(IComparer), а также в ряде других мест в классах коллекций. Главное преимущество применения интерфейса IComparer заключается в том, что сортировке подлежат объекты тех классов, в которых интерфейс IComparable не реализуется.

Интерфейс IComparer<T> является обобщенным вариантом интерфейса IComparer. В нем определяется приведенный ниже обобщенный вариант метода Compare():



В этом методе сравниваются объекты х и у и возвращается нулевое значение, если значения сравниваемых объектов равны; положительное — если значение объекта х больше, чем у объекта у; и отрицательное — если значение объекта х меньше, чем у объекта у.

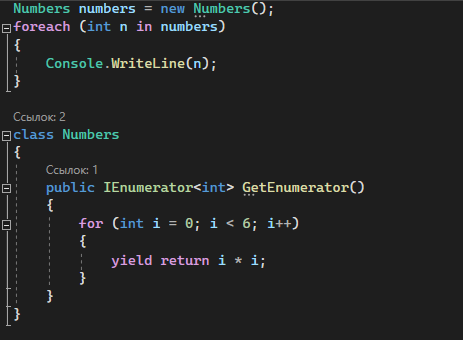
**Итераторы и оператор yield**

Итератор по сути представляет блок кода, который использует оператор yield для перебора набора значений. Данный блок кода может представлять тело метода, оператора или блок get в свойствах.

Итератор использует две специальных инструкции:

* yield return: определяет возвращаемый элемент
* yield break: указывает, что последовательность больше не имеет элементов

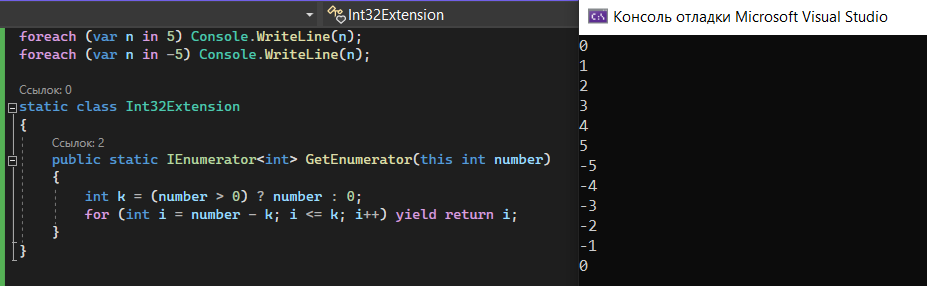
Рассмотрим небольшой пример:

****

В классе Numbers метод GetEnumerator() фактически представляет итератор. С помощью оператора yield return возвращается некоторое значение (в данном случае квадрат числа).

В программе с помощью цикла foreach мы можем перебрать объект Numbers как обычную коллекцию. При получении каждого элемента в цикле foreach будет срабатывать оператор yield return, который будет возвращать один элемент и запоминать текущую позицию.

Благодаря итераторам мы можем пойти дальше и легко реализовать перебор числа в цикле foreach:



В данном случае итератор реализован как метод расширения для типа int или System.Int32. В методе итератора фактически возвращаем все целочисленные значения от 0 до текущего числа.