Массивы и другие линейные структуры данных

Описание массива и его инициализация

Массивы: *одномерные* (single-dimensional), многомерные *прямоугольные* (multidimensional), *массивы массивов — невыровненные* (jagged).

Описание:

```
char[] a;
int[,] b;
```

Создание с неявной инициализацией:

```
a = new char[5];
b = new int[3, 4];
```

Совмещение описания и инициализации:

```
char[] a = new char[5];
int[,] b = new int[3, 4];
```

Явная инициализация:

```
a = new char[] {'A', 'B', 'C', 'D', 'E'};
b = new int[,] {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};
```

Повторное создание:

```
a = new char[5];
…
a = new char[10];
```

Создание «на лету»:

```
M(new char[] {'A', 'B', 'C'});
```

Описание и неявная инициализация невыровненного двумерного целочисленного массива `d` с элементами $d_{0,0}$, $d_{1,0}$, $d_{1,1}$, $d_{2,0}$, $d_{2,1}$, $d_{2,2}$:

```
int[][] d = new int[3][];
d[0] = new int[1];
d[1] = new int[2];
d[2] = new int[3];
d[1][0] = 5;
```

Класс Array

Свойства и экземплярные методы

```
int Length { get; }
int Rank { get; }
int GetLength(int dimension);
int GetUpperBound(int dimension);
object Clone();
void CopyTo(Array a, int start);
```

Классовые методы

```
static void Clear(Array a, int start, int count);
static void Copy(Array src, Array dest, int count);
static void Copy(Array src, int srcStart, Array dest, int destStart, int count);
static void Reverse(Array a[, int start, int count]);
static void Sort(Array a[, Array b][, int start, int count][, System.Collections.IComparer
   cmp]);
static int IndexOf(Array a, object value[, int start[, int count]]);
static int LastIndexOf(Array a, object value[, int start[, int count]]);
```

Динамический массив («список на базе массива»): классы ArrayList и List<T>

Динамические массивы, в отличие от обычных, могут изменять свой размер после создания (с сохранением своего прежнего содержимого): `System.Collections.ArrayList` и `System.Collections.Generic.List<T>`

Наборы свойств и методов классов ArrayList и List<T> совпадают. Ниже описаны члены класса List<T>. Всюду, где в описании члена класса List<T> используется тип T[] и T, в описании аналогичного члена класса ArrayList надо использовать тип Array и object соответственно.

Свойства

```
int Capacity { get; set; }
int Count { get; }
T [int index] { get; set; }
```

Конструкторы и другие способы создания динамического массива

```
List<T>([int capacity]);
List<T>(IEnumerable<T> c);
object Clone();
static List<T> Repeat(T value, int count);
```

Совместимость объектов Array и List<T>

Обычные массивы и динамические массивы List<T> не совместимы по присваиванию. Для преобразования обычного массива к объекту List<T> необходимо воспользоваться соответствующим конструктором класса List<T>.

Обратное преобразование:

```
T[] ToArray(); // в ArrayList имеет тип object[]
void CopyTo(T[] array[, int arrayStart]);
void CopyTo(int start, T[] array, int arrayStart, int count);
```

В классе ArrayList есть вариант метода ToArray с дополнительным параметром. Пример его использования:

```
double[] a = (double[])d.ToArray(typeof(double));
```

Преобразование динамического массива

```
void Add(T value); // в ArrayList имеет тип int
void Insert(int index, T value);
void AddRange(IEnumerable<T> c);
void InsertRange(int start, IEnumerable<T> c);
void Remove(T value);
void RemoveAt(int index);
void RemoveRange(int start, int count);
void Clear();
void TrimToSize();
void Reverse([int start, int count]);
void Sort([[int start, int count,] System.Collections.IComparer cmp]);
```

Поиск в динамическом массиве

```
bool Contains(T value);
int IndexOf(T value[, int start[, int count]]);
int LastIndexOf(T value[, int start[, int count]]);
```

Перебор элементов в цикле foreach

Признаком допустимости использования цикла foreach для объекта является наличие у этого объекта интерфейса IEnumerable.

Пример (a имеет тип List<int>):

```
foreach (int v in a)
    s += v;
```

Без foreach:

```
for (int i = 0; i < a.Count; i++)
    s += a[i];
```

Двусвязный список LinkedList<T>

Хранит данные в виде цепочки связанных узлов и имеет средства, позволяющие проходить эту цепочку как в прямом, так и в обратном направлении. Благодаря подобной организации своих данных двусвязный список может очень быстро выполнять операции вставки и удаления узлов. Однако у него отсутствуют средства прямого доступа к узлам (по индексу).

Свойства LinkedList<T>: Count типа int, First и Last типа LinkedListNode<T> (все только для чтения). Если список пуст, то First и Last равны null. **Конструктор**: без параметров или с параметром-коллекцией. Тип LinkedListNode<T> имеет четыре свойства: List, Next, Previous и Value (доступно для записи только последнее). Есть конструктор с одним параметром Value.

Методы AddFirst, AddLast, AddBefore, AddAfter (все имеют по два перегруженных варианта – как AddFirst и AddAfter):

```
LinkedListNode<T> AddFirst(T value)
AddFirst(LinkedListNode<T> newNode)
LinkedListNode<T> AddAfter(LinkedListNode<T> node, T value)
AddAfter(LinkedListNode<T> node, LinkedListNode<T> newNode)
```

Добавляемый узел newNode не должен принадлежать какому-либо списку (т. е. его свойство List должно быть равно null).

```
bool Remove(T value)
Remove(LinkedListNode<T> node)
RemoveFirst()
RemoveLast()
Clear()
CopyTo(T[] arr, int arrStart)
bool Contains(T value)
LinkedListNode<T> Find(T value)
LinkedListNode<T> FindLast(T value)
```

Множества HashSet<T> и SortedSet<T>

Имеют почти одинаковый набор свойств и методов; отличаются способом хранения элементов. Класс HashSet использует для хранения хеш-таблицу, что позволяет осуществлять быстрый поиск элемента в множестве. Класс SortedSet для этих же целей хранит свои элементы в отсортированном виде (что позволяет использовать быстрый алгоритм бинарного поиска).

Свойства: Comparer, Count. Дополнительно SortedSet<T> имеет свойства: Min, Max (все только для чтения). Для пустого множества Min и Max возвращают нулевые значения типа T. **Конструкторы**: без параметров, с параметром-коллекцией, с параметром comparer типа IEqualityComparer<T> для сравнения элементов множества.

Методы:

```
bool Add(T item)
bool Remove(T item)
int RemoveWhere( T => bool)
Clear()
bool Contains(T item)
bool SetEquals(IEnumerable<T> other)
bool IsProperSubsetOf(IEnumerable<T> other)
bool IsSubsetOf(IEnumerable<T> other)
bool IsProperSupersetOf(IEnumerable<T> other)
bool IsSupersetOf(IEnumerable<T> other)
bool Overlaps(IEnumerable<T> other)
```

Коллекция other может содержать одинаковые элементы; порядок элементов тоже не играет роли.

```
UnionWith(IEnumerable<T> other)
ExceptWith(IEnumerable<T> other)
IntersectWith(IEnumerable<T> other)
SymmetricExceptionWith(IEnumerable<T> other)
```

Методы изменяют множество, для которого были вызваны.

```
CopyTo(T[] arr [, int arrStart])
```

В случае SortedSet элементы записываются в возрастающем порядке, а в случае HashSet – в произвольном.

Методы SortedSet (в возвращаемый поддиапазон включаются границы):

```
SortedSet<T> GetViewBetween(T minValue, T maxValue)
IEnumerable<T> Reverse()
```

Словарь (ассоциативный массив) Dictionary<TKey, TVal>

Хранит данные в виде набора пар «ключ–значение» (key–value), причем для доступа к значению value пары (key, value) из словаря d можно использовать операцию индексирования по ключу: d[key]. Особое поведение операции d[key] = val;

Свойства: Count, Keys, Values, Comparer (все только для чтения). Keys и Values тоже имеют свойство Count.

Пары представляются в виде типа KeyValuePair<TKey, TVal> со свойствами Key и Value, доступными только для чтения. Имеет конструктор с параметрами (key, value).

Конструктор: без параметров, с параметром capacity, с параметром comparer, с параметром dict типа Dictionary<TKey, TVal>.

Три варианта цикла foreach: по словарю d (параметр цикла имеет тип KeyValuePair<TKey, TVal>), по d.Keys (параметр имеет тип TKey), по d.Values (параметр имеет тип TVal).

Методы:

```
Add(TKey key, TVal value)
bool Remove(TKey key)
Clear()
bool ContainsKey(TKey key)
bool ContainsValue(TVal value)
bool TryGetValue(TKey key, TVal: TKey; out TVal value)
```

Метод коллекций Keys и Values:

```
CopyTo(T[] arr, int arrStart)
```