Коллекции

- В C# под ***коллекцией*** понимается некоторая группа объектов.

- Коллекции упрощают реализацию многих задач программирования, предлагая уже готовые решения для построения структур данных.

- Все коллекции разработаны на основе четко определенных интерфейсов, поэтому стандартизируют способ обработки группы объектов.

- Среда .NET Framework поддерживает три основных типа коллекций:
  - **общего назначения,**
  - **специализированные**
  - **ориентированные на побитовую организацию данных**.

Типы коллекций

- ***Коллекции общего назначения*** определены в пространстве имен **System.Collection** и реализуют структуры данных:
  - стеки,
  - очереди,
  - динамические массивы,
  - словари (хеш-таблицы, предназначенные для хранения пар ключ/значение),
  - отсортированный список для хранения пар ключ/значение. Коллекции общего назначения работают с данными типа object, поэтому их можно использовать для хранения данных любого типа.

Типы коллекций

- ***Коллекции специального назначения*** определены в пространстве имен **System.Collection.Specialized** и ориентированы на обработку данных конкретного типа или на обработку данных уникальным способом (существуют специализированные коллекции, предназначенные только для обработки строк).

- ***Коллекция, ориентированная на побитовую организацию данных*** единственная определена в пространстве имен **System.Collection**, служит для хранения групп битов и поддерживает такой набор операций, который не характерен для коллекций других типов.

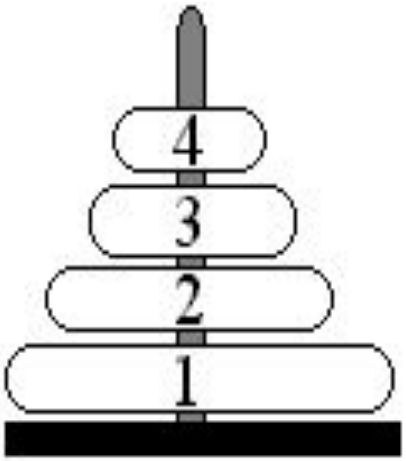
Коллекции общего назначения

Класс	*Описание*
Stack	Стек – частный случай однонаправленного списка, действующий по принципу: последним пришел – первым вышел
Queue	Очередь – частный случай однонаправленного списка, действующего по принципу:первым пришел – первым вышел
ArrayList	Динамический массив, т.е. массив который при необходимости может увеличивать свой размер
Hashtable	Хеш-таблица для пар ключ/значение
SortedList	Отсортированный список пар ключ/значение

Абстрактный тип данных

- Абстрактный тип данных (АТД) ***список*** – это последовательность элементов a_1, a_2,…, a_n ($n \geq 0$) одного типа.
- Количество элементов *n* называется ***длиной списка***.
- Если $n > 0$, то a_1 называется ***первым элементом списка***, а ***an – последним элементом списка***.
- В случае $n = 0$ имеем ***пустой список***, который не содержит элементов.
- Элементы списка линейно упорядочены в соответствии с их позицией в списке.
- Список называется ***однонаправленным***, если каждый элемент списка содержит ссылку на следующий элемент.
- Если каждый элемент списка содержит две ссылки (одну на следующий элемент в списке, вторую – на предыдущий элемент), то такой список называется ***двунаправленным*** (двусвязным).
- А если последний элемент связать указателем с первым, то получится ***кольцевой*** список.

Класс Stack



- АТД стек – это частный случай однонаправленного списка, добавление элементов в который и выборка элементов из которого выполняются с одного конца, называемого вершиной стека (головой – head).

- При выборке элемент исключается из стека.

- Стек реализует принцип обслуживания LIFO (last in – fist out, последним пришел – первым вышел).

класс Stack

- реализует интерфейсы ICollection, IEnumerable и ICloneable.
- Stack – это динамическая коллекция, размер которой изменяется.
- В классе Stack определены следующие конструкторы:

public Stack(); //создает пустой стек, начальная вместимость которого равна 10

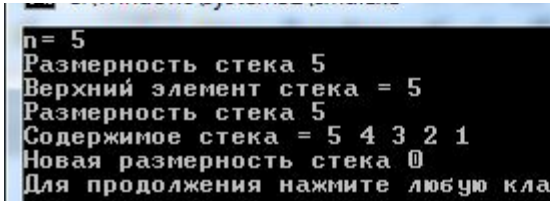
public Stack(int capacity); // создает пустой стек, начальная вместимость которого равна capacity

public Stack(ICollection c); //создает стек, который содержит элементы коллекции, заданной параметром c

Кроме методов, определенных в интерфейсах, реализуемых классом Stack, в классе определены собственные методы:

Метод	Описание
public virtual bool Contains(object v)	Возвращает значение true, если объект v содержится в вызывающем стеке, в противном случае возвращает значение false.
public virtual void Clear()	Устанавливает свойство Count равным нулю, тем самым очищая стек.
public virtual object Peek()	Возвращает элемент, расположенный в вершине стека, но не извлекая его из стека
public virtual object Pop()	Возвращает элемент, расположенный в вершине стека, и извлекает его из стека
public virtual void Push(object v)	Помещает объект v в стек
public virtual object[] ToArray()	Возвращает массив, который содержит копии элементов вызывающего стека

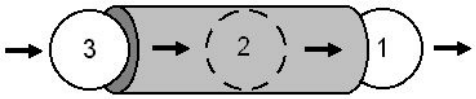
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1{
//Для заданного значения n запишем в стек все числа от 1 до n, а затем
    извлечем из стека
class Program{
    public static void Main(){
    Console.Write("n= ");
    int n=int.Parse(Console.ReadLine());
    Stack intStack = new Stack();
      for (int i = 1; i <= n; i++)
      intStack.Push(i);
    Console.WriteLine("Размерность стека " + intStack.Count);
    Console.WriteLine("Верхний элемент стека = " + intStack.Peek());
    Console.WriteLine("Размерность стека " + intStack.Count);
    Console.Write("Содержимое стека = ");
while (intStack.Count != 0)
      Console.Write("{0} ", intStack.Pop());
      Console.WriteLine("\nНовая размерность стека " + intStack.Count);}
}   }
```



```
// В текстовом файле содержится математическое выражение. Проверить баланс круглых
    скобок в данном выражении.
class Program{
    public static void Main(){
      StreamReader fileIn=new StreamReader("t.txt");
      string line=fileIn.ReadToEnd();        fileIn.Close();
      Stack skobki=new Stack();
      bool flag=true;
      //проверяем баланс скобок
    for ( int i=0; i<line.Length;i++)  {
//если текущий символ открывающаяся скобка, то помещаем ее в стек
    if (line[i] == '(') skobki.Push(i);
       else if (line[i] == ')')      //если текущий символ закрывающаяся скобка, то

//если стек пустой, то для закрывающейся скобки не хватает парной открывающейся
    {if (skobki.Count == 0)
    { flag = false;          Console.WriteLine("Возможно в позиции " + i + "лишняя ) скобка"); }
       else skobki.Pop(); }      }        //иначе извлекаем парную скобку
//если после просмотра строки стек оказался пустым, то скобки сбалансированы
    if (skobki.Count == 0) { if (flag)Console.WriteLine("скобки сбалансированы"); }
      else   //иначе баланс скобок нарушен
      {       Console.Write("Возможно лишняя ( скобка в позиции:");
    while (skobki.Count != 0)
      {       Console.Write("{0} ", (int)skobki.Pop());   }
    Console.WriteLine();}      }
```

Класс Queue



- Очередь - частный случай однонаправленного списка, добавление элементов в который выполняется в один конец (хвост), а выборка производится с другого конца (головы).
- Другие операции с очередью не определены.
- При выборке элемент исключается из очереди.
- Очередь реализует принцип обслуживания FIFO (fist in – fist out, первым пришел – первым вышел).

класс Queue

- реализует интерфейсы **ICollection**, **IEnumerable** и **ICloneable**.
- **Queue** – это динамическая коллекция, размер которой изменяется.
- При необходимости увеличение вместимости очереди происходит с коэффициентом роста по умолчанию равным 2.0.
- В классе Queue определены следующие конструкторы:

public Queue(); //создает пустую очередь, начальная вместимость которой равна 32

public Queue (int capacity); // создает пустую очередь, начальная вместимость которой равна capacity

//создает пустую очередь, начальная вместимость которой равна capacity, и коэффициент роста
//устанавливается параметром n
public Queue (int capacity, float n);

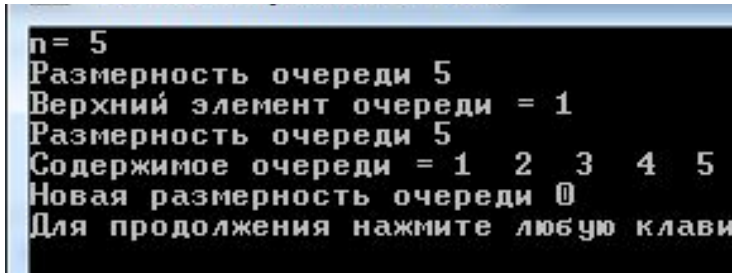
//создает очередь, которая содержит элементы коллекции, заданной параметром c, и аналогичной
//вместимостью
public Queue (ICollection c);

Кроме методов, определенных в интерфейсах, реализуемых классом Queue, в классе определены собственные методы:

Метод	Описание
public virtual bool Contains (object v)	Возвращает значение true, если объект v содержится в вызывающей очереди, в противном случае возвращает значение false
public virtual void clear ()	Устанавливает свойство Count равным нулю, тем самым очищая очередь
public virtual object Dequeue ()	Возвращает объект из начала вызывающей очереди, удаляя его из очереди
public virtual object Peek ()	Возвращает объект из начала вызывающей очереди, не удаляя его из очереди
public virtual void Enqueue(object v)	Добавляет объект v в конец очереди
public virtual object [ ] ToArray ()	Возвращает массив, который содержит копии элементов из вызывающей очереди
public virtual void TrimToSizeO	Устанавливает свойство Capacity равным значению свойства Count

```
//Для заданного значения n запишем в очередь все числа от 1 до n,
  а затем извлечем их из очереди:
class Program {
  public static void Main()          {
        Console.Write("n= ");
        int n=int.Parse(Console.ReadLine());
        Queue intQ = new Queue();
        for (int i = 1; i <= n; i++)      intQ.Enqueue(i);
        Console.WriteLine("Размерность очереди " + intQ.Count);

  Console.WriteLine("Верхний элемент очереди = " + intQ.Peek());
        Console.WriteLine("Размерность очереди " + intQ.Count);
            Console.Write("Содержимое очереди = " );
        while (intQ.Count!=0)
            Console.Write("{0}  ", intQ.Dequeue());
  Console.WriteLine("\nНовая размерность очереди " + intQ.Count);
  }
}
```



```
/*В текстовом файле записана информация о людях (фамилия, имя, отчество, возраст, вес через пробел).
Вывести на экран вначале информацию о людях младше 40 лет, а затем информацию о всех остальных*/
    class Program{
      public struct one //структура для хранения данных об одном человеке
      {public string f;  public string i;  public string o; public int age; public float massa;}
      public static void Main(){
            StreamReader fileIn = new StreamReader("t.txt",Encoding.GetEncoding(1251));
            string line; Queue people = new Queue();  one a;
            Console.WriteLine("ВОЗРАСТ МЕНЕЕ 40 ЛЕТ");
            while ((line = fileIn.ReadLine()) != null) //читаем до конца файла
            {       string [] temp = line.Split(' ');//разбиваем строку на составные элементы
                    //заполняем структуру
                    a.f = temp[0];        a.i = temp[1];        a.o = temp[2];
                    a.age = int.Parse(temp[3]);       a.massa = float.Parse(temp[4]);
// если возраст меньше 40 лет, то выводим данные на экран, иначе помещаем их в очередь для временного
    хранения
            if (a.age<40)   Console.WriteLine(a.f + "\t"+ a.i + "\t"+ a.o + "\t"+a.age + "\t" + a.massa);
                        else people.Enqueue(a);          }
            fileIn.Close();   Console.WriteLine("ВОЗРАСТ 40 ЛЕТ И СТАРШЕ");
                    while (people.Count != 0) //извлекаем из очереди данные
                    {       a = (one)people.Dequeue();
            Console.WriteLine(a.f + "\t"+ a.i + "\t"+ a.o + "\t"+a.age + "\t" + a.massa);}   }       }
```

Класс ArrayList

- предназначен для поддержки динамических массивов, которые при необходимости могут увеличиваться или сокращаться.
- Объект класса ArrayList представляет собой массив переменной длины, элементами которого являются объектные ссылки.
- Любой объект класса ArrayList создается с некоторым начальным размером.
- При превышении этого размера коллекция автоматически удваивается.
- В случае удаления объектов массив можно сократить.

Класс ArrayList

- реализует интерфейсы **ICollection**, **IList**, **IEnumerable** и **ICloneable**.

- В классе ArrayList определены следующие конструкторы:

//создает пустой массив с максимальной емкостью равной 16 элементам, при текущей размерности 0

public ArrayList()

public ArrayList(int capacity) //создает массив с заданной емкостью capacity, при текущей размерности 0

public ArrayList(ICollection c) //строит массив, который инициализируется элементами коллекции c

Собственные методы класса ArrayList

Метод	Описание
public virtual void AddRange (ICollection c)	Добавляет элементы из коллекции *c* в конец вызывающей коллекции
public virtual int BinarySearch (object v)	В вызывающей отсортированной коллекции выполняет поиск значения, заданного параметром v. Возвращает индекс найденного элемента. Если искомое значение не обнаружено, возвращает отрицательное значение.
public virtual int BinarySearch (object v, IComparer comp)	В вызывающей отсортированной коллекции выполняет поиск значения, заданного параметром v, на основе метода сравнения объектов, заданного параметром comp. Возвращает индекс найденного элемента. Если искомое значение не обнаружено, возвращает отрицательное значение.
public virtual int BinarySearch (int startIdx, int count, object v, IComparer comp)	В вызывающей отсортированной коллекции выполняет поиск значения, заданного параметром v, на основе метода сравнения объектов, заданного параметром comp. Поиск начинается с элемента, индекс которого равен значению startIdx, и включает count элементов. Метод возвращает индекс найденного элемента. Если искомое значение не обнаружено, возвращает отрицательное значение.

Собственные методы класса ArrayList

Метод	Описание
public virtual void CopyTo(Array ar, int startIdx)	Копирует содержимое вызывающей коллекции, начиная с элемента, индекс которого равен значению startIdx, в массив, заданный параметром ar. Приемный массив должен быть одномерным и совместимым по типу с элементами коллекции.
public virtual void CopyTo(int srcldx, Array ar, int destIdx, int count)	Копирует count элементов вызывающей коллекции, начиная с элемента, индекс которого равен значению srcIdx, в массив, заданный параметром ar, начиная с элемента, индекс которого равен значению destIdx. Приемный массив должен быть одномерным и совместимым по типу с элементами коллекции
public virtual ArrayList GetRange(int idx, int ount)	Возвращает часть вызывающей коллекции типа ArrayList. Диапазон возвращаемой коллекции начинается с индекса idx и включает count элементов. Возвращаемый объект ссылается на те же элементы, что и вызывающий объект
public static ArrayList FixedSize(ArrayList ar)	Превращает коллекцию ar в ArrayList-массив с фиксированным размером и возвращает результат
public virtual void InsertRange(int startldx, Icollection c)	Вставляет элементы коллекции, заданной параметром c, в вызывающую коллекцию, начиная с индекса, заданного параметром startIdx
public virtual int LastIndexOf(object v)	Возвращает индекс последнего вхождения объекта v в вызывающей коллекции. Если искомый объект не обнаружен, возвращает отрицательное значение

Собственные методы класса ArrayList

Метод	Описание
public static ArrayList Readonly(ArrayList ar)	Превращает коллекцию ar в ArrayList-массив, предназначенный только для чтения
public virtual void ReraoveRange (int idx, int count)	Удаляет count элементов из вызывающей коллекции, начиная с элемента, индекс которого равен значению idx
public virtual void Reverse()	Располагает элементы вызывающей коллекции в обратном порядке
public virtual void Reverse(int startldx, int count)	Располагает в обратном порядке count элементов вызывающей коллекции, начиная с индекса startldx
public virtual void SetRange(int startldx, ICollection c)	Заменяет элементы вызывающей коллекции, начиная с индекса startldx, элементами коллекции, заданной параметром с
public virtual void Sort()	Сортирует коллекцию по возрастанию

Собственные методы класса ArrayList

Метод	Описание
public virtual void Sort(IComparer comp)	Сортирует вызывающую коллекцию на основе метода сравнения объектов, заданного параметром comp. Если параметр comp имеет нулевое значение, для каждого объекта используется стандартный метод сравнения
Public virtual void Sort (int startidx, int endidx, icomparer comp)	Сортирует часть вызывающей коллекции на основе метода сравнения объектов, заданного параметром comp. Сортировка начинается с индекса startidx и заканчивается индексом endidx. Если параметр comp имеет нулевое значение, для каждого объекта используется стандартный метод сравнения
public virtual object [ ] ToArray ()	Возвращает массив, который содержит копии элементов вызывающего объекта
public virtual Array ToArray (Type type)	Возвращает массив, который содержит копии элементов вызывающего объекта. Тип элементов в этом массиве задается параметром type
public virtual void TrimToSize()	Устанавливает свойство Capacity равным значению свойства Count

Свойство Capacity

- позволяет узнать или установить емкость вызывающего динамического массива типа ArrayList.

- Емкость представляет собой количество элементов, которые можно сохранить в ArrayList-массиве без его увеличения.

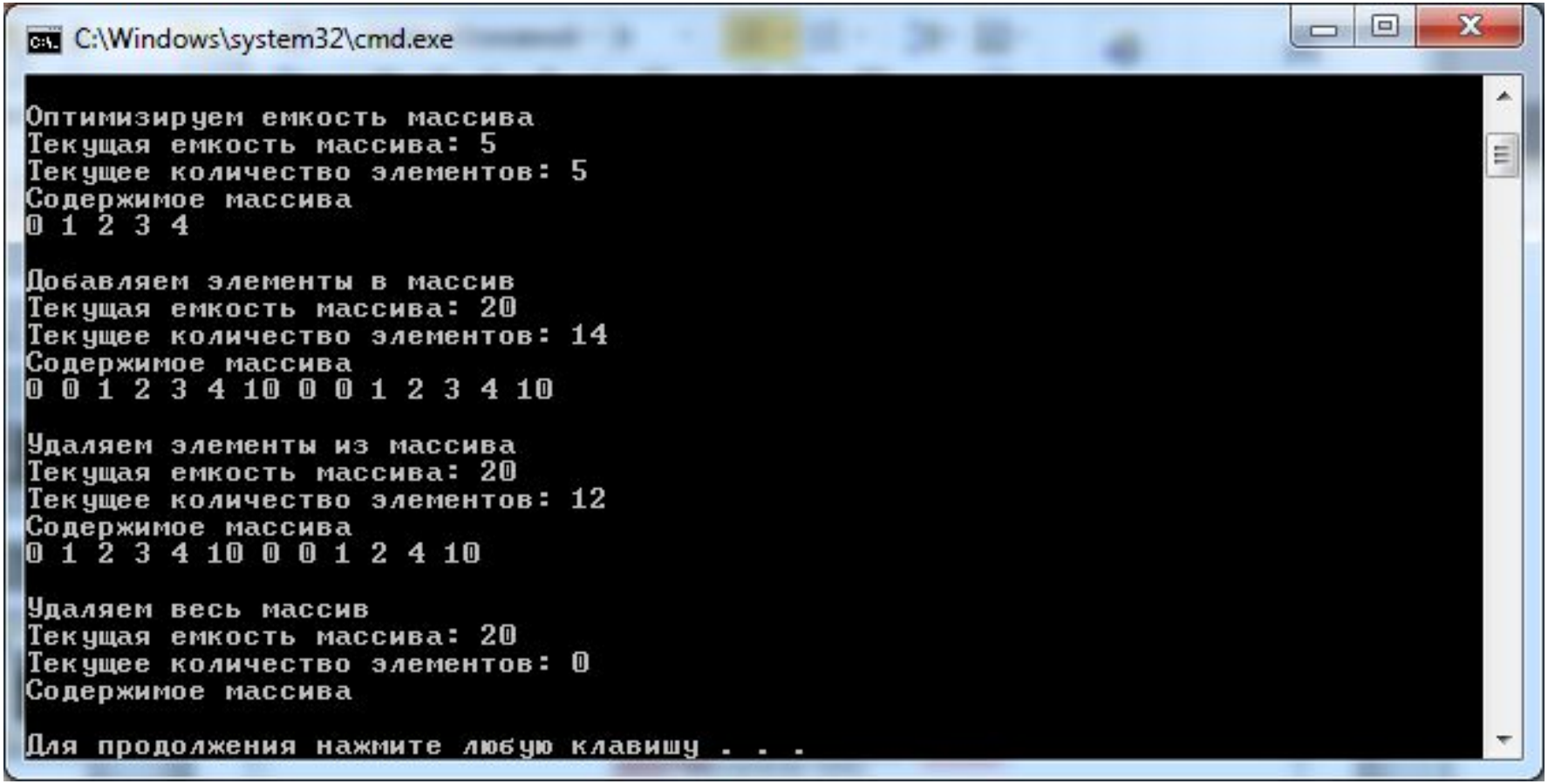
- Если вам заранее известно, сколько элементов должно содержаться в ArrayList-массиве, то размерность массива можно установить используя свойство Capacity (экономим системные ресурсы.)

- Если нужно уменьшить размер ArrayList-массива, то путем установки свойства Capacity можно сделать его меньшим. Но устанавливаемое значение не должно быть меньше значения свойства Count (иначе будет сгенерировано исключение ArgumentOutOfRangeException).

- Чтобы сделать емкость ArrayList-массива равной действительному количеству элементов, хранимых в нем в данный момент, установите свойство Capacity равным свойству Count. Того же эффекта можно добиться, вызвав метод TrimToSize ().

```
//использование динамического массива
    class Program{
      static void ArrayPrint(string s, ArrayList a)
      {       Console.WriteLine(s);
              foreach (int i in a)          Console.Write(i + " ");
              Console.WriteLine();
      }
      static void Main(string[] args)  {
              ArrayList myArray = new ArrayList();
              Console.WriteLine("Начальная емкость массива: " + myArray.Capacity);
              Console.WriteLine("Начальное количество элементов: " + myArray.Count);
              Console.WriteLine("\nДобавили 5 цифр");
              for (int i = 0; i < 5; i++) myArray.Add(i);
              Console.WriteLine("Текущая емкость массива: " + myArray.Capacity);
              Console.WriteLine("Текущее количество элементов: " + myArray.Count);
              ArrayPrint("Содержимое массива", myArray);
                    Console.WriteLine("\nОптимизируем емкость массива");
              myArray.Capacity=myArray.Count;
              Console.WriteLine("Текущая емкость массива: " + myArray.Capacity);
```

```
Console.WriteLine("Текущее количество элементов: " + myArray.Count);
ArrayPrint("Содержимое массива", myArray);
Console.WriteLine("\nДобавляем элементы в массив");
myArray.Add(10);
myArray.Insert(1, 0);
myArray.AddRange(myArray);
Console.WriteLine("Текущая емкость массива: " + myArray.Capacity);
Console.WriteLine("Текущее количество элементов: " + myArray.Count);
ArrayPrint("Содержимое массива", myArray);
Console.WriteLine("\nУдаляем элементы из массива");
myArray.Remove(0);
myArray.RemoveAt(10);
Console.WriteLine("Текущая емкость массива: " + myArray.Capacity);
Console.WriteLine("Текущее количество элементов: " + myArray.Count);
ArrayPrint("Содержимое массива", myArray);
Console.WriteLine("\nУдаляем весь массив");
myArray.Clear();
Console.WriteLine("Текущая емкость массива: " + myArray.Capacity);
Console.WriteLine("Текущее количество элементов: " + myArray.Count);
ArrayPrint("Содержимое массива", myArray);  }       }
```


C:\Windows\system32\cmd.exe
Оптимизируем емкость массива
Текущая емкость массива: 5
Текущее количество элементов: 5
Содержимое массива
0 1 2 3 4
Добавляем элементы в массив
Текущая емкость массива: 20
Текущее количество элементов: 14
Содержимое массива
0 0 1 2 3 4 10 0 0 1 2 3 4 10
Удаляем элементы из массива
Текущая емкость массива: 20
Текущее количество элементов: 12
Содержимое массива
0 1 2 3 4 10 0 0 1 2 4 10
Удаляем весь массив
Текущая емкость массива: 20
Текущее количество элементов: 0
Содержимое массива
Для продолжения нажмите любую клавишу . . .

```
/*В текстовом файле записана информация о людях (фамилия, имя, отчество, возраст, вес
  через пробел). Вывести на экран информацию о людях, отсортированную по возрасту*/
class Program{
  public struct one //структура для хранения данных об одном человеке
  {  public string f;   public string I;  public string o;     public int age;   public float massa; }
  public class SortByAge : IComparer  //реализация  стандартного интерфейса
  {       int IComparer.Compare(object x, object y)   //переопределение метода Compare
          {       one t1 = (one)x;   one t2 = (one)y;
          if (t1.age > t2.age) return 1;    if (t1.age < t2.age) return -1; return 0;    }       }
  static void ArrayPrint(string s, ArrayList a) {
  Console.WriteLine(s);
  foreach (one x in a)       Console.WriteLine(x.f + "\t"+ x.i + "\t"+ x.o + "\t"+x.age + "\t" + x.massa);       }
  static void Main(string[] args) {
  StreamReader fileIn = new StreamReader("t.txt",Encoding.GetEncoding(1251));
  string line;  one a;        ArrayList people = new ArrayList();  string[] temp = new string[3];
          while ((line=fileIn.ReadLine())!=null) //цикл для организации обработки файла
          {temp = line.Split(' ');    a.f = temp[0];       a.i = temp[1];a.o = temp[2];
          a.age = int.Parse(temp[3]);   a.massa = float.Parse(temp[4]);  people.Add(a);   }
                  fileIn.Close();
                  ArrayPrint("Исходные данные: ", people);
                  people.Sort(new Program.SortByAge());    //вызов сортировки
                  ArrayPrint("Отсортированные данные: ", people);        }       }
```

Класс Hashtable

- Предназначен для создания коллекции, в которой для хранения объектов используется хеш-таблица.

- Суть хеширования: для определения уникального значения, которое называется хеш-кодом, используется информационное содержимое соответствующего ему ключа.

- Хеш-код затем используется в качестве индекса, по которому в таблице отыскиваются данные, соответствующие этому ключу.

- Преобразование ключа в хеш-код выполняется автоматически, т.е. сам хеш-код мы не видим.

- Преимущество хеширования —оно позволяет сокращать время выполнения операций: поиск, считывание и запись данных (даже для больших объемов информации).

Класс Hashtable

- реализует стандартные интерфейсы **IDictionary**, **ICollection**, **IEnumerable**, **ISerializable**, **IDeserializationCallback** и **ICloneable**.

- Размер хеш-таблицы может динамически изменяться. Размер таблицы увеличивается тогда, когда количество элементов превышает значение, равное произведению вместимости таблицы и ее коэффициента заполнения, который может принимать значение на интервале от 0,1 до 1,0 (по умолчанию установлен коэффициент равный 1,0).

  В классе Hashtable определено несколько конструкторов:

public Hashtable() //создает пустую хеш-таблицу

// строит хеш-таблиц, которая инициализируется элементами коллекции *c*
public Hashtable(IDictionary c)

public Hashtable(int capacity) //создает хеш-таблицу с вместимостью *capacity*

//создает хеш-таблицу вместимостью capacity и коэффициентом заполнения n
public Hashtable(int capacity, float n)

Собственные методы класса Hashtable

Метод	Описание
public virtual bool ContainsKey (object k)	Возвращает значение true , если в вызывающей хеш-таблице содержится ключ, заданный параметром к. В противном случае возвращает значение false
public virtual bool ContainsValue (object v)	Возвращает значение true, если в вызывающей хеш-таблице содержится значение, заданное параметром v. В противном случае возвращает значение false
public virtual IDictionaryEnumerator GetEnumerator()	Возвращает для вызывающей хеш-таблицы нумератор типа IDictionaryEnumerator

два собственных public-свойства класса Hashtable

//позволяет получить коллекцию ключей

public virtual ICollection Keys { get; }

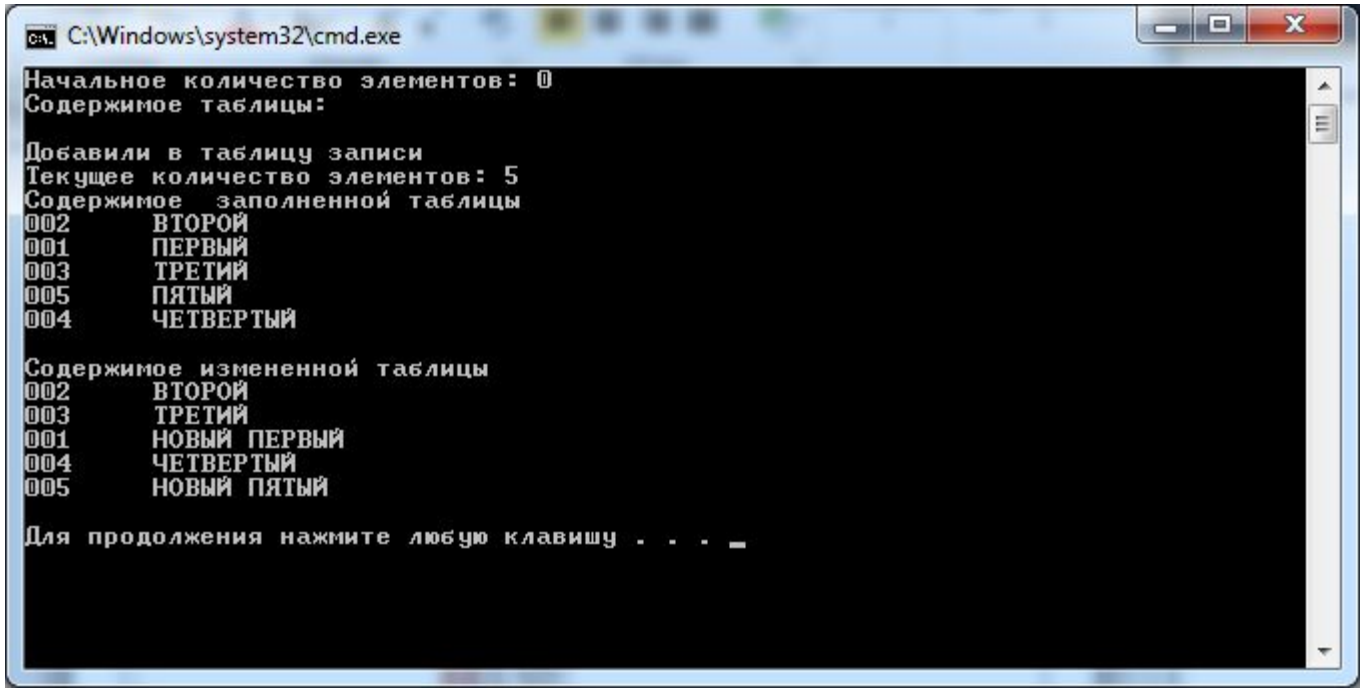
//позволяет получить коллекцию значений

public virtual ICollection Values { get; }

- Для добавления элемента в хеш-таблицу необходимо вызвать метод **Add()**, который принимает два отдельных аргумента: ключ и значение.

- Хеш-таблица не гарантирует сохранения порядка элементов, т.к хеширование обычно не применяется к отсортированным таблицам.

```
//простые операции с хеш-таблицей
    class Program {
            static void printTab(string s, Hashtable a)
     {      Console.WriteLine(s);
            ICollection key = a.Keys; //Прочитали все ключи
            foreach (string i in key)  //использование ключа для получения значения
            {       Console.WriteLine(i+"\t"+a[i]);  }
            Console.WriteLine();
     }
     static void Main(string[] args)
     {      Hashtable tab = new Hashtable();
            Console.WriteLine("Начальное количество элементов: " + tab.Count);
            printTab("Содержимое таблицы: ", tab);
            Console.WriteLine("Добавили в таблицу записи");
            tab.Add("001","ПЕРВЫЙ");    tab.Add("002","ВТОРОЙ");
            tab.Add("003","ТРЕТИЙ");     tab.Add("004", "ЧЕТВЕРТЫЙ");
            tab.Add("005", "ПЯТЫЙ");
            Console.WriteLine("Текущее количество элементов: " + tab.Count);
            printTab("Содержимое  заполненной таблицы", tab);
            tab["005"] = "НОВЫЙ ПЯТЫЙ"; tab["001"] = "НОВЫЙ ПЕРВЫЙ";
            printTab("Содержимое измененной таблицы", tab);
     }              }
```


C:\Windows\system32\cmd.exe
Начальное количество элементов: 0
Содержимое таблицы:
Добавили в таблицу записи
Текущее количество элементов: 5
Содержимое заполненной таблицы
002 ВТОРОЙ
001 ПЕРВЫЙ
003 ТРЕТИЙ
005 ПЯТЫЙ
004 ЧЕТВЕРТЫЙ
Содержимое измененной таблицы
002 ВТОРОЙ
003 ТРЕТИЙ
001 НОВЫЙ ПЕРВЫЙ
004 ЧЕТВЕРТЫЙ
005 НОВЫЙ ПЯТЫЙ
Для продолжения нажмите любую клавишу . . .

```
/*записная книжка- добавлять, удалять телефоны, поиск номера телефона по фамилии и фамилии
    по номеру телефона*/
class Program     {
  static void printTab(string s, Hashtable a){Console.WriteLine(s);ICollection key = a.Keys; //Прочитали все
    ключи
     foreach (string i in key)
          { Console.WriteLine(i + "\t" + a[i]);}}//использование ключа для получения значения
     static void Main(string[] args){ StreamReader fileIn = new StreamReader("t.txt",Encoding.GetEncoding(1251));
     string line;   Hashtable people = new Hashtable();
     while ((line = fileIn.ReadLine()) != null) //цикл для организации обработки файла
          {string [] temp = line.Split(' ');  people.Add(temp[0],temp[1]);       }
   fileIn.Close();          printTab("Исх. данные: ", people);
   Console.WriteLine("Введите № телефона");    line = Console.ReadLine();
          if (people.ContainsKey(line)) Console.WriteLine(line + "\t" + people[line]);
          else{Console.WriteLine("Такого номера нет в записной книжке.\nВведите фамилию: ");
               string line2=Console.ReadLine();      people.Add(line,line2); }
   printTab("Исходные данные: ", people); Console.WriteLine("Введите фамилию для удаления");
   line = Console.ReadLine();
          if (people.ContainsValue(line))        {ICollection key =people.Keys; //Прочитали все ключи
          Console.WriteLine(line);          string del="";
          foreach (string i in key) //использование ключа для получения значения
          if (string.Compare((string)people[i], line) == 0)   {del = i;       break;         }
    Console.WriteLine(del + "\t" + people[del] + "- данные удалены!!!");         people.Remove(del);
   printTab("Измененные данные: ", people);   }
          else Console.WriteLine("Такого абонента в записной книжке нет ");        } }
```