Структуры данных

Контейнерные классы

Работа с файлами

Абстрактные структуры данных

- *Массив*

  конечная совокупность однотипных величин. Занимает непрерывную область памяти и предоставляет прямой (произвольный) доступ к элементам по индексу.

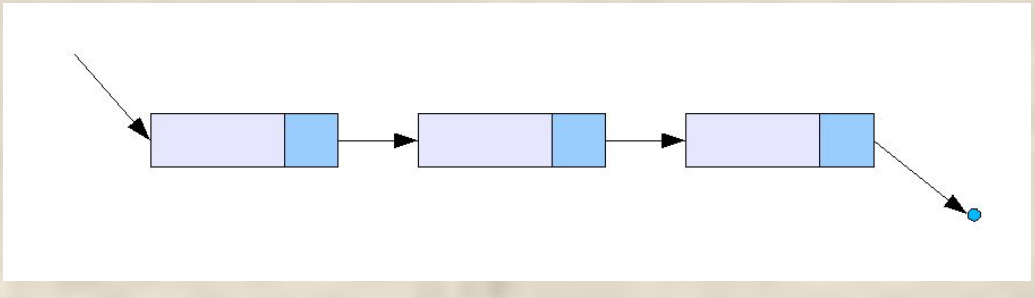
- *Линейный список*
  - *Стек*
  - *Очередь*
- *Дерево*
  - *Бинарное дерево*
- *Хеш-таблица (ассоциативный массив, словарь)*
- *Граф*
- *Множество*

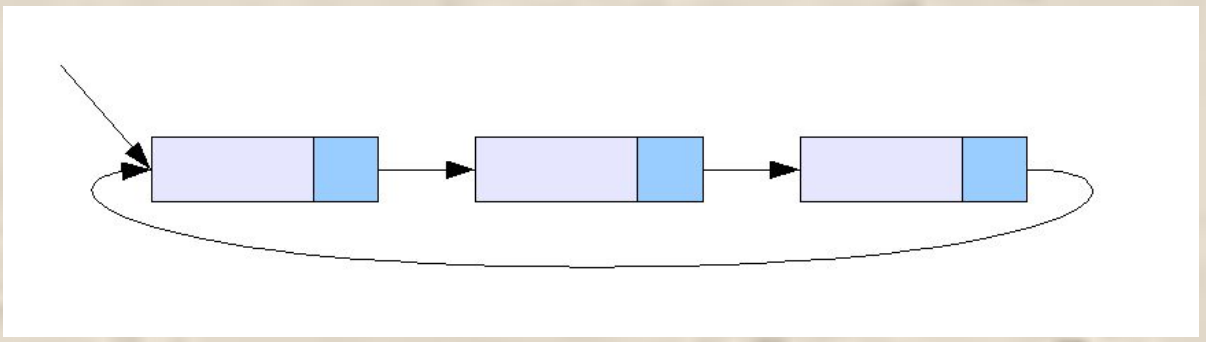
Линейный список

- В *списке* каждый элемент связан со следующим и, возможно, с предыдущим. Количество элементов в списке может изменяться в процессе работы программы.
- Каждый элемент списка содержит *ключ*, идентифицирующий этот элемент.
- Виды списков:

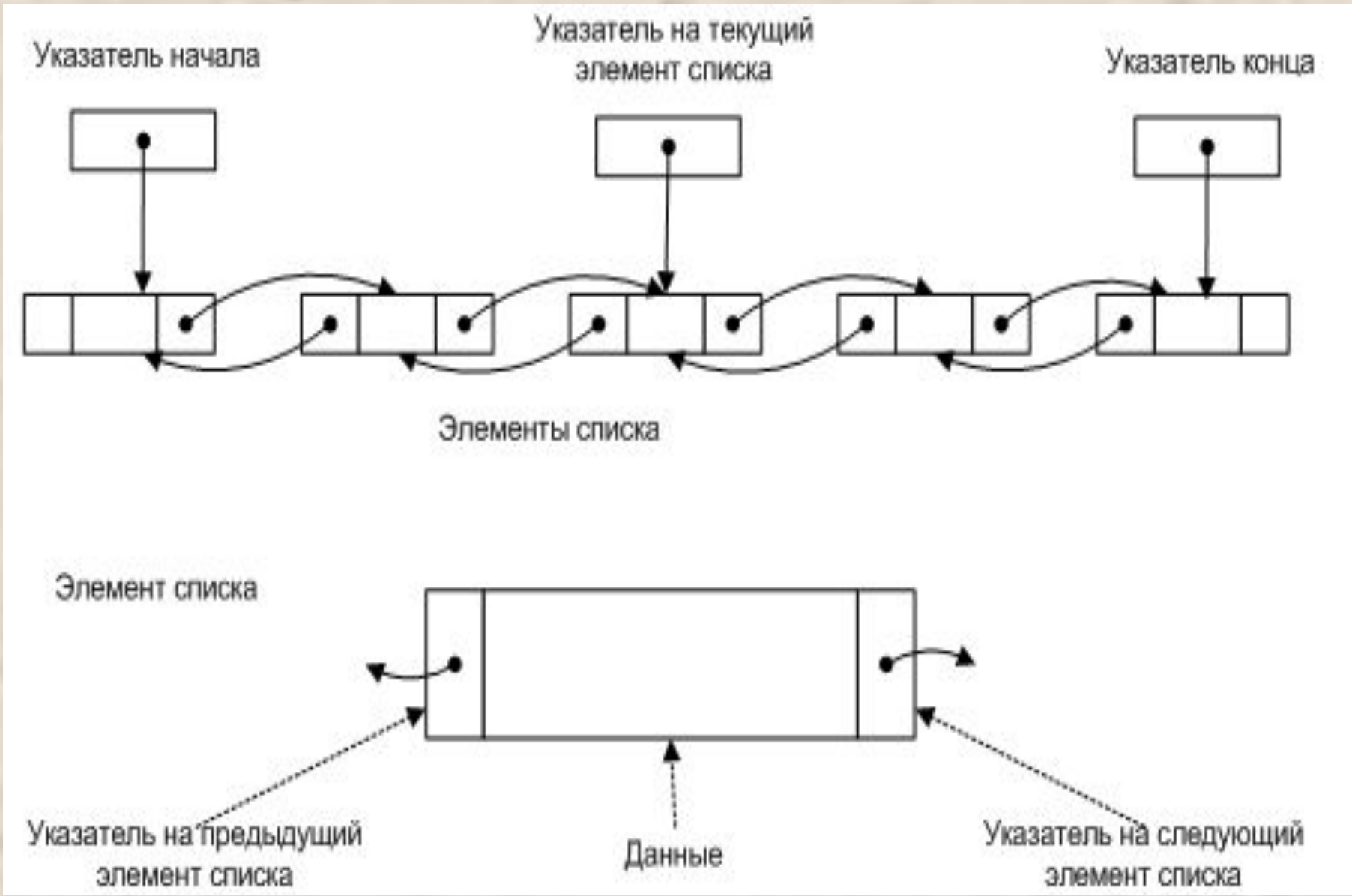
односвязные ⟵ ⟶ кольцевые

двусвязные



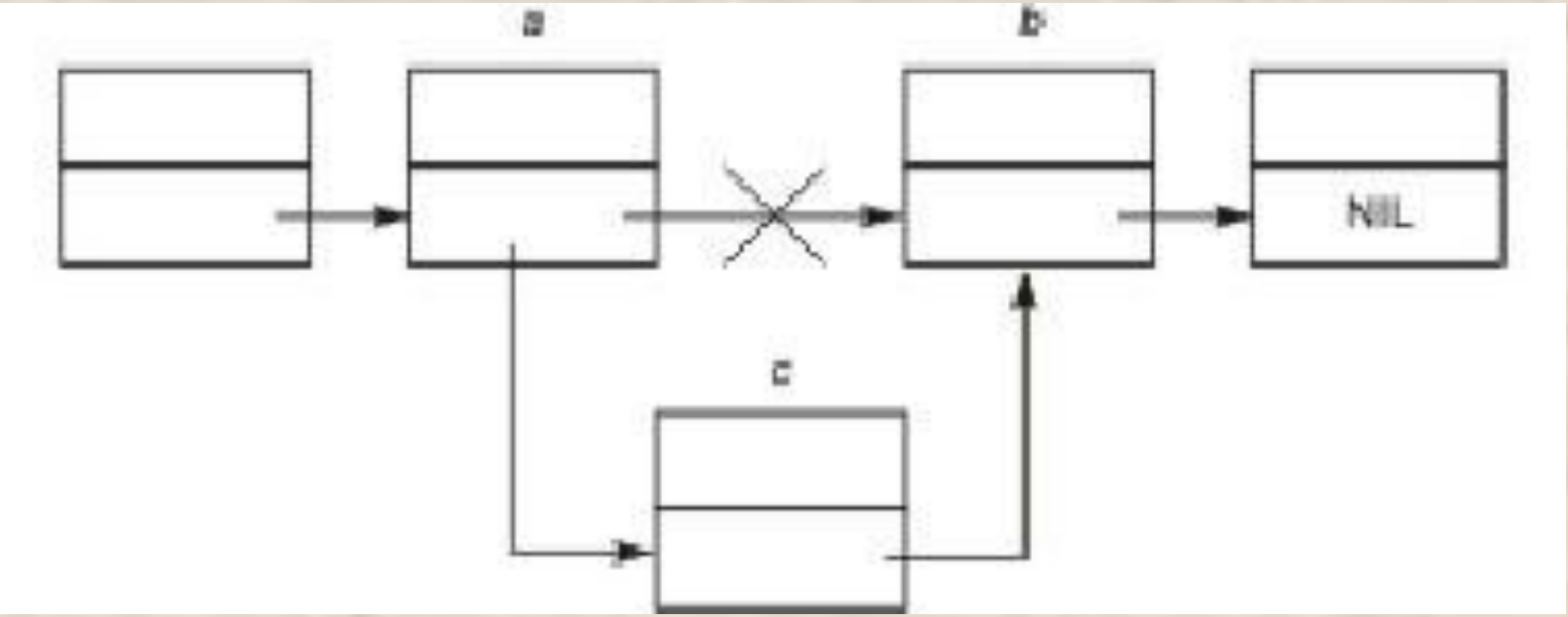


Двусвязный список



Преимущества списка перед массивом

– Простая вставка элемента:



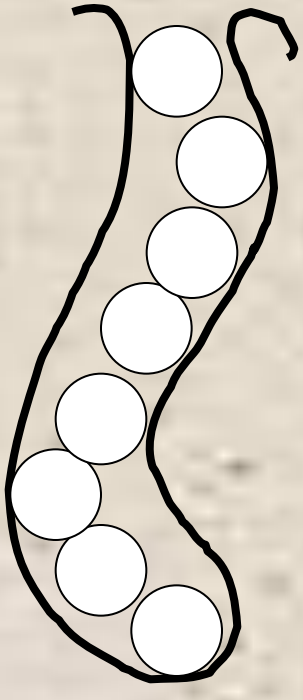
- Удаление элемента
- Сортировка элементов (??)

Стек

Стек — частный случай однонаправленного списка, добавление элементов в который и выборка из которого выполняются с одного конца, называемого вершиной стека (стек реализует принцип обслуживания **LIFO**).

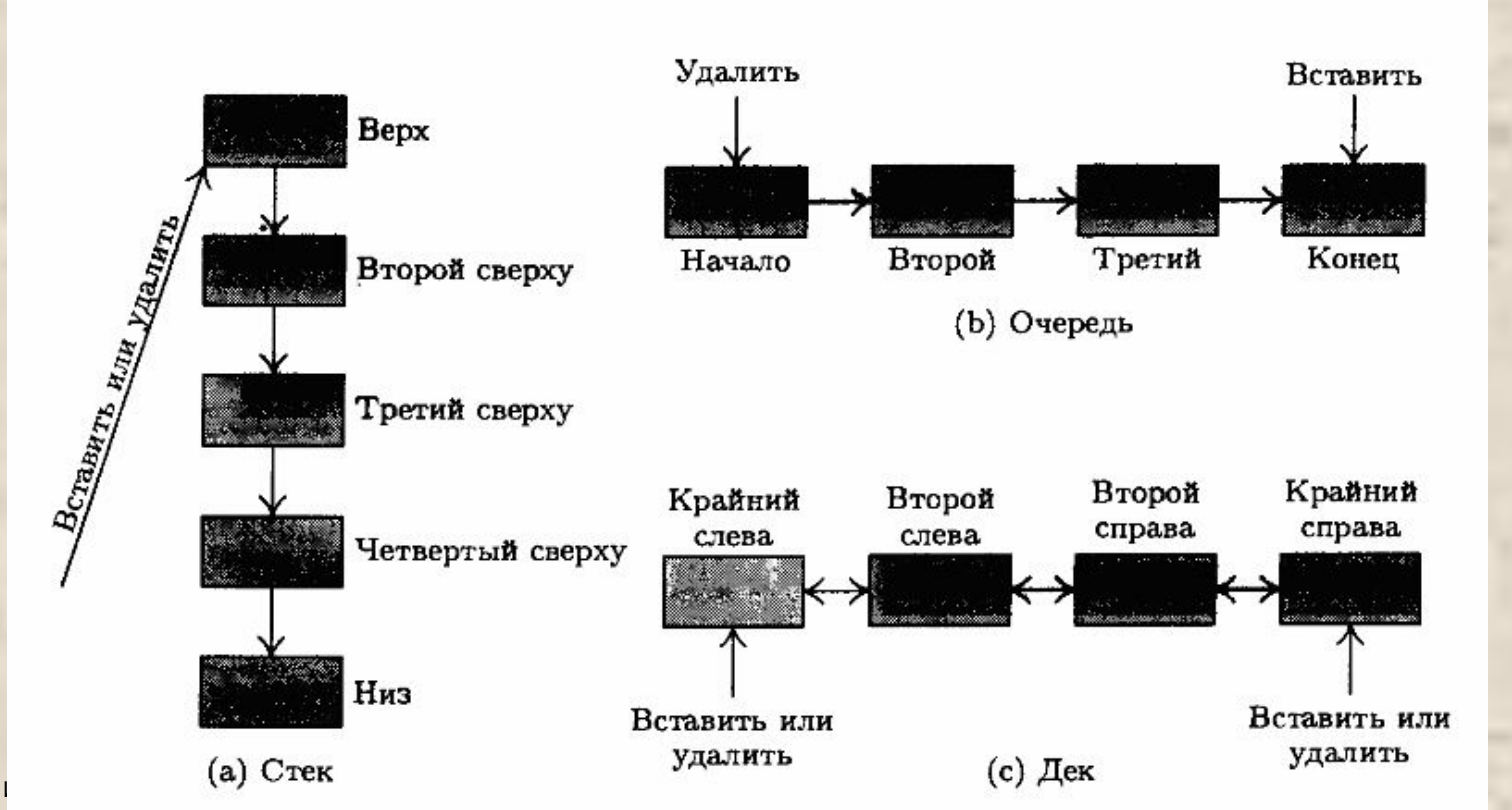
Другие операции со стеком не определены.

При выборке элемент исключается из стека.



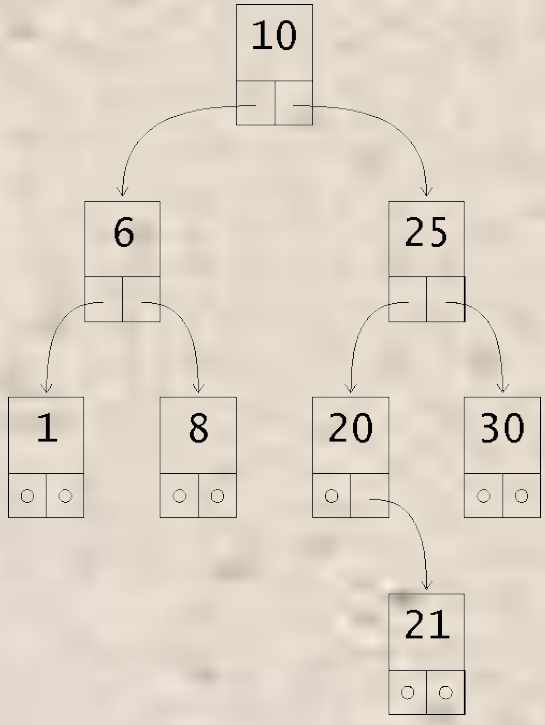
Очередь

- *Очередь* — частный случай однонаправленного списка, добавление элементов в который выполняется в один конец, а выборка — из другого конца (очередь реализует принцип обслуживания **FIFO**). Другие операции с очередью не определены.
- При выборке элемент исключается из очереди.



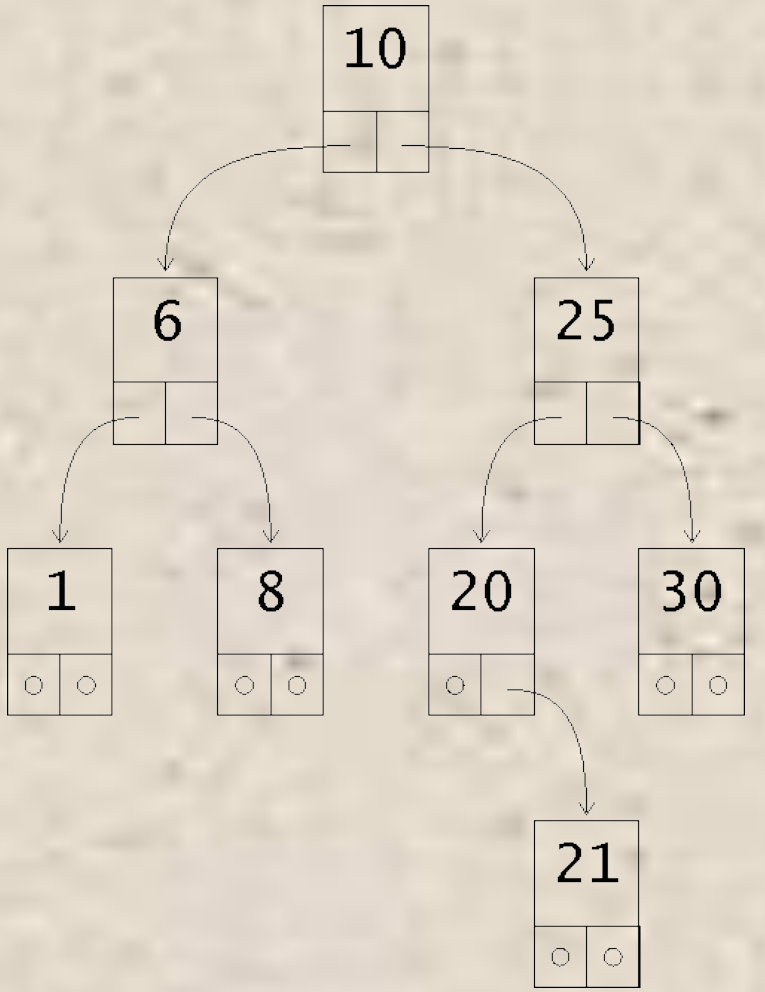
Бинарное дерево

- *Бинарное дерево* — динамическая структура данных, состоящая из узлов, каждый из которых содержит, помимо данных, не более двух ссылок на различные бинарные поддеревья.
- На каждый узел имеется ровно одна ссылка. Начальный узел называется *корнем* дерева.
- Узел, не имеющий поддеревьев, называется *листом*. Исходящие узлы называются *предками*, входящие — *потомками*.
- *Высота дерева* определяется количеством уровней, на которых располагаются его узлы.



Дерево поиска

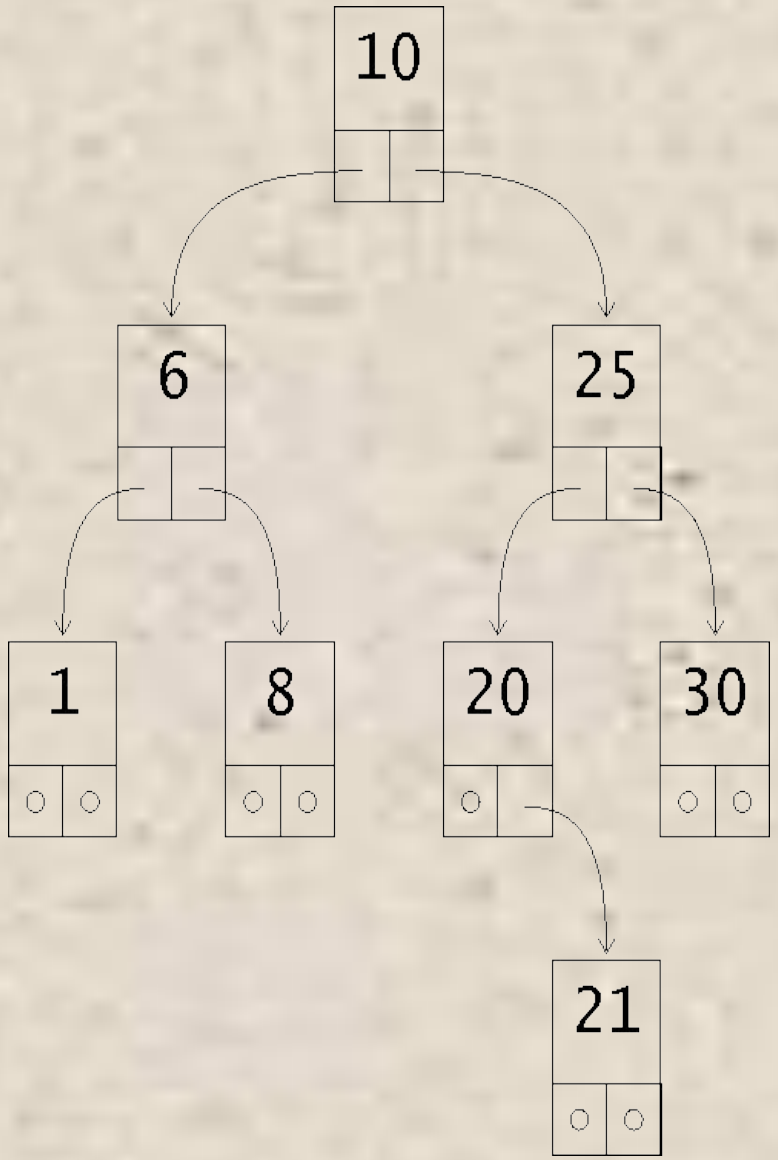
- Если дерево организовано таким образом, что для каждого узла все ключи его левого поддерева меньше ключа этого узла, а все ключи его правого поддерева — больше, оно называется *деревом поиска*.
- Одинаковые ключи не допускаются.
- В дереве поиска можно найти элемент по ключу, двигаясь от корня и переходя на левое или правое поддерево в зависимости от значения ключа в каждом узле.



Обход дерева

```
procedure print_tree( дерево );
begin
    print_tree( левое_поддерево )
    посещение корня
    print_tree( правое_поддерево )
end;
```

1 6 8 10 20 21 25 30



Хеш-таблица

- *Хеш-таблица (ассоциативный массив, словарь)* — массив, доступ к элементам которого осуществляется не по номеру, а по ключу (т.е. это таблица, состоящая из пар «ключ-значение»)
  - рус-англ-словарь:

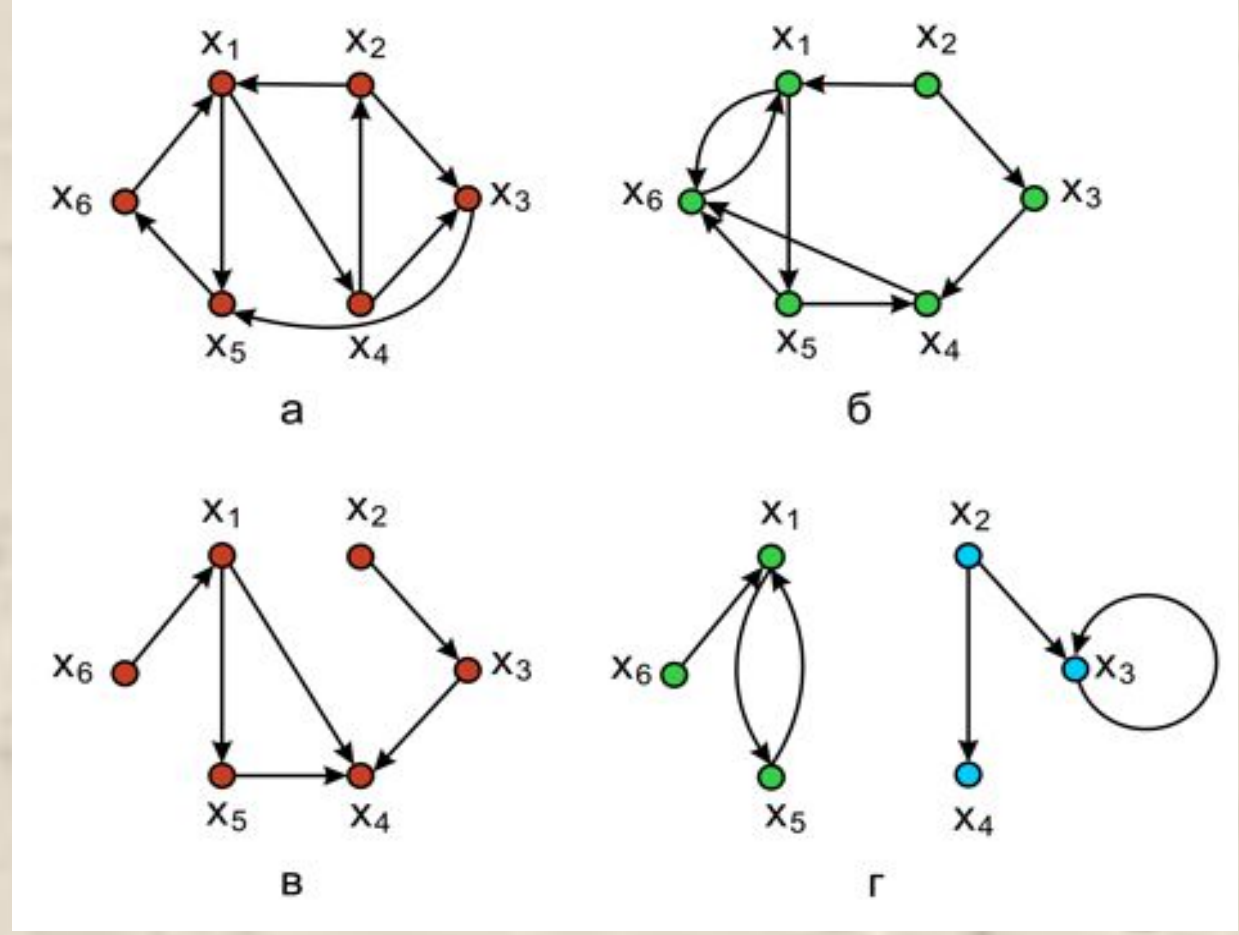
    {кукла} -> doll

Ключ	**Значение**
boy	мальчик
girl	девочка
dog	собачка

Хеш-таблица эффективно реализует операцию поиска значения по ключу. Ключ преобразуется в число (*хэш-код*), которое используется для быстрого нахождения нужного значения в хеш-таблице.

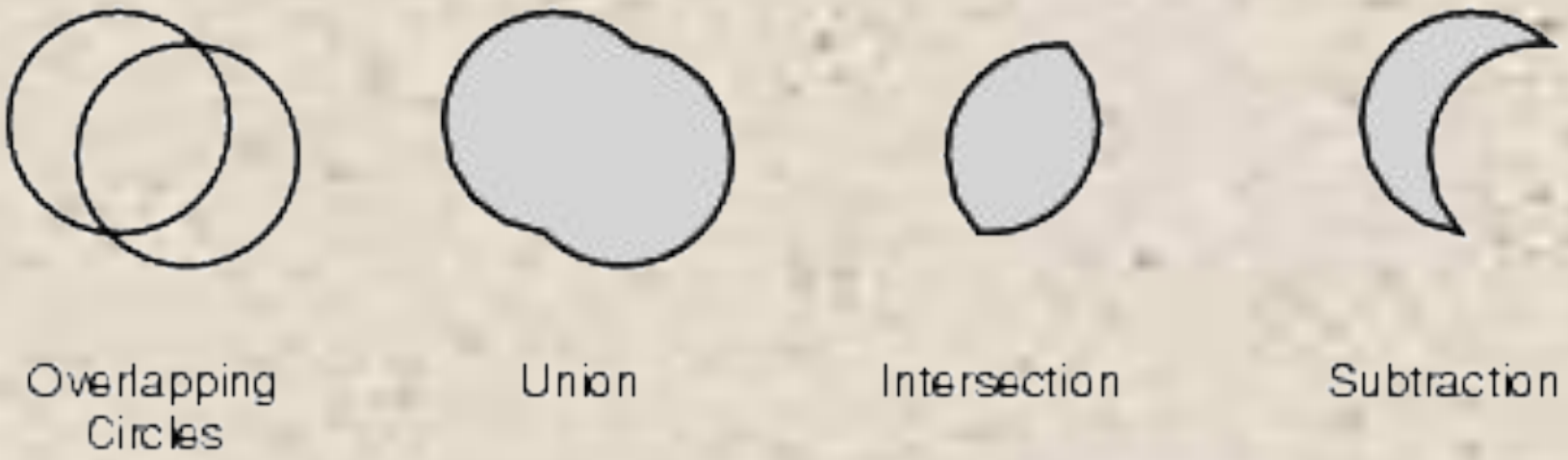
Граф

- *Граф* — совокупность узлов и ребер, соединяющих различные узлы. Множество реальных практических задач можно описать в терминах графов, что делает их структурой данных, часто используемой при написании программ.



Множество

- *Множество* — неупорядоченная совокупность элементов. Для множеств определены операции:
  - проверки принадлежности элемента множеству
  - включения и исключения элемента
  - объединения, пересечения и вычитания множеств.



- Все эти структуры данных называются *абстрактными*, поскольку в них не задается реализация допустимых операций.

Контейнеры http://msdn.microsoft.com/ru-ru/library/ybcx56wz.aspx?ppud=4

- *Контейнер (коллекция)* - стандартный класс, реализующий абстрактную структуру данных.
- Для каждого типа коллекции определены методы работы с ее элементами, не зависящие от конкретного типа хранимых данных.
- Использование коллекций позволяет сократить сроки разработки программ и повысить их надежность.
- Каждый вид коллекции поддерживает свой набор операций над данными, и быстродействие этих операций может быть разным.
- Выбор вида коллекции зависит от того, что требуется делать с данными в программе и какие требования предъявляются к ее быстродействию.
- В библиотеке .NET определено множество стандартных контейнеров.
- Основные пространства имен, в которых они описаны — System.Collections, System.Collections.Specialized и System.Collections.Generic

System.Collections

ArrayList	Массив, динамически изменяющий свой размер
BitArray	Компактный массив для хранения битовых значений
Hashtable	Хэш-таблица
Queue	Очередь
SortedList	Коллекция, отсортированная по ключам. Доступ к элементам — по ключу или по индексу
Stack	Стек

Параметризованные коллекции (классы-прототипы, generics)

- классы, имеющие типы данных в качестве параметров

Класс-прототип (версия 2.0)	**Обычный класс**
Dictionary<K,T>	HashTable
LinkedList<T>	—
List<T>	ArrayList
Queue<T>	Queue
SortedDictionary<K,T>	SortedList
Stack<T>	Stack

Выбор класса коллекции

- Нужен ли последовательный список, элемент которого обычно удаляется сразу после извлечения его значения (Queue, Stack)
- Нужен ли доступ к элементам в определенном порядке (FIFO - Queue, LIFO - Stack) или в произвольным порядке (LinkedList)
- Необходимо ли иметь доступ к каждому элементу по индексу? (ArrayList, StringCollection, List, …)
- Будет ли каждый элемент содержать только одно значение, сочетание из одного ключа и одного значения или сочетание из одного ключа и нескольких значений?
- Нужна ли возможность отсортировать элементы в порядке, отличном от порядка их поступления? (HashTable, SortedList, SortedDictionary, …)
- Необходимы ли быстрый поиск и извлечение данных? (Dictionary)
- Нужна ли коллекция только для хранения строк? (StringCollection, StringDictionary, *типиз_коллекция*<String>)

Пример использования класса List

```
using System;
using System.Collections.Generic;
namespace ConsoleApplication1{
class Program {
        static void Main()  {
       List<int> lint = new List<int>(); // массив из целых
            lint.Add( 5 ); lint.Add( 1 ); lint.Add( 3 );
            lint.Sort();
            int a = lint[2];          Console.WriteLine( a );
            foreach ( int x in lint ) Console.Write( x + " ");
// массив из монстров:
       List<Monster> stado = new List<Monster>();
            stado.Add( new Monster( "Monia" ) ); /*  …  */
            foreach ( Monster x in stado ) x.Passport();
}}}
```

Пример использования класса Dictionary: формирование частотного словаря

```
class Program {
    static void Main() {
    StreamReader f = new StreamReader( @"d:\C#\text.txt" );
    string s = f.ReadToEnd();
    char[] separators = { '.', ' ', ',', '!' };
    List<string> words = new List<string>( s.Split(separators) );
    Dictionary<string, int> map = new Dictionary<string, int>();
    foreach ( string w in words ) {
          if ( map.ContainsKey( w ) ) map[w]++;
          else  map[w] = 1;                  // слово встретилось впервые
          }
    foreach ( string w in map.Keys )
          Console.WriteLine( "{0}\t{1}", w, map[w] );
}}
```

Обобщенные методы (generic methods)

```
// Пример: сортировка выбором
class Program {
        static void Sort<T> ( ref T[] a )                          // 1
            where T : IComparable<T>                               // 2
        {  T buf;
            int n = a.Length;
            for ( int i = 0; i < n - 1; ++i ) {
                int im = i;
                for ( int j = i + 1; j < n; ++j )
                    if ( a[j].CompareTo(a[im]) < 0 ) im = j;       // 3
                buf = a[i]; a[i] = a[im]; a[im] = buf;
            }
        }
```

Продолжение примера

```
static void Main() {
        int[] a = { 1, 6, 4, 2, 7, 5, 3 };
        Sort<int>( ref a );                                    // 4
        foreach ( int elem in a ) Console.WriteLine( elem );

        double[] b = { 1.1, 5.2, 5.21, 2, 7, 6, 3 };
        Sort( ref b );                                         // 5
        foreach ( double elem in b ) Console.WriteLine( elem );

        string[] s = { "qwe", "qwer", "df", "asd" };
        Sort( ref s );                                         // 6
        foreach ( string elem in s ) Console.WriteLine( elem );
    }
}
```

Преимущества generics

Параметризованные типы и методы позволяют:

- описывать способы хранения и алгоритмы обработки данных независимо от типов данных;
- выполнять контроль типов во время компиляции программы.

Применение generics увеличивает надежность программ и уменьшает сроки их разработки.

Организация справки MSDN

Для каждого элемента:

- Имя
- Назначение
- Пространство имен, сборка
- Синтаксис (Syntax)
- Описание (Remarks)
- Примеры (Examples)
- Иерархия наследования, платформы, версия, …
- Ссылки на родственную информацию (See also)

Пример справки для List <T>

List <T> Class

Represents a strongly typed list of objects that can be accessed by index. Provides methods to search, sort, and manipulate lists.

Namespace: System.Collections.Generic

Syntax:

```
[SerializableAttribute]
public class List<T> : IList<T>, ICollection<T>,
    IEnumerable<T>, IList, ICollection, IEnumerable
```

Type Parameters:

- T - The type of elements in the list.

Remarks

- The List <T > class is the generic equivalent of the ArrayListThe List <T > class is the generic equivalent of the ArrayList class. It implements the IList <T > generic interface using an array whose size is dynamically increased as required.
- The List <T > class uses both an equality comparer and an ordering comparer.
- Methods such as ContainsMethods such as Contains, IndexOfMethods such as Contains, IndexOf, LastIndexOfMethods such as Contains, IndexOf, LastIndexOf, and RemoveMethods such as Contains, IndexOf, LastIndexOf, and Remove use an equality comparer for the list elements. The default equality comparer for type T is determined as follows. If type T implements the IEquatable <T >Methods such as Contains, IndexOf, LastIndexOf, and Remove use an equality comparer for the list elements. The default equality comparer for type T is determined as follows. If type T implements the IEquatable <T > generic interface, then the equality comparer is the Equals(T)Methods such as Contains, IndexOf, LastIndexOf, and Remove use an equality comparer for the list elements. The default equality comparer for type T is determined as follows. If type T implements the IEquatable <T > generic interface, then the equality comparer is the Equals(T) method of that interface; otherwise, the default equality comparer is Object .Equals(Object) .
- Methods such as BinarySearchMethods such as BinarySearch and SortMethods such as BinarySearch and Sort use an ordering comparer for the list elements. The default comparer for type T is determined as follows. If type T implements the IComparable <T >Methods such as BinarySearch and Sort use an ordering comparer for the list elements. The default comparer for type T is determined as follows. If type T implements the IComparable <T > generic interface, then the default comparer is the CompareTo(T)Methods such as BinarySearch and Sort use an ordering comparer for the list elements. The default comparer for type T is determined as follows. If type T implements the IComparable <T > generic interface, then the default comparer is the CompareTo(T) method of that interface; otherwise, if type T implements the nongeneric IComparableMethods such as BinarySearch and Sort use an ordering comparer for the list elements. The default comparer for type T is determined as follows. If type T implements the IComparable <T > generic interface, then the default

Examples

- The following code example demonstrates several properties and methods of the List <T > generic class of type string. (For an example of a List <T > of complex types, see the Contains method.)
- The default constructor is used to create a list of strings with the default capacity. The CapacityThe default constructor is used to create a list of strings with the default capacity. The Capacity property is displayed and then the AddThe default constructor is used to create a list of strings with the default capacity. The Capacity property is displayed and then the Add method is used to add several items. The items are listed, and the CapacityThe default constructor is used to create a list of strings with the default capacity. The Capacity property is displayed and then the Add method is used to add several items. The items are listed, and the Capacity property is displayed again, along with the Count property, to show that the capacity has been increased as needed.
- The ContainsThe Contains method is used to test for the presence of an item in the list, the Insert method is used to insert a new item in the middle of the list, and the contents of the list are displayed again.
- The default ItemThe default Item property (the indexer in C#) is used to retrieve an item, the RemoveThe default Item property (the indexer in C#) is used to retrieve an item, the Remove method is used to remove the first instance of the duplicate item added earlier, and the contents are displayed again. The Remove method always removes the first instance it encounters.
- The TrimExcessThe TrimExcess method is used to reduce the capacity to match the count, and the CapacityThe TrimExcess method is used to reduce the capacity to match the count, and the Capacity and Count properties are

```
using System;
using System.Collections.Generic;
public class Example {
public static void Main() {
   List<string> dinosaurs = new List<string>();
   Console.WriteLine("\nCapacity: {0}", dinosaurs.Capacity);
   dinosaurs.Add("Tyrannosaurus");
   dinosaurs.Add("Amargasaurus");
   foreach(string dinosaur in dinosaurs) { Console.WriteLine(dinosaur); }
   Console.WriteLine("Count: {0}", dinosaurs.Count);
   Console.WriteLine("\nContains(\"Deinonychus\"): {0}",
    dinosaurs.Contains("Deinonychus"));
   Console.WriteLine("\nInsert(2, \"Compsognathus\")"); dinosaurs.Insert(2,
    "Compsognathus");
   foreach(string dinosaur in dinosaurs) { Console.WriteLine(dinosaur); }
…
```

Результат работы примера

```
/* This code example produces the following output:
Capacity: 0
Tyrannosaurus
Amargasaurus
Capacity: 8
Count: 5
Contains("Deinonychus"): True
Insert(2, "Compsognathus")
…
```

See Also

Reference

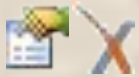
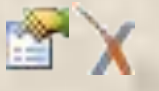
List <T > Members

System.Collections.Generic Namespace

IList

List <T > Members

Properties

	Name	Description
	Capacity	Gets or sets the total number of elements the internal data structure can hold without resizing.
	Count	Gets the number of elements actually contained in the List <T >.
	Item	Gets or sets the element at the specified index.

Методы

- Methods

ForEach, GetEnumerator, GetHashCode, GetRange, GetType, IndexOf(T), IndexOf(T, Int32), IndexOf(T, Int32, Int32), Insert, InsertRange, LastIndexOf(T), LastIndexOf(T, Int32), LastIndexOf(T, Int32, Int32), MemberwiseClone, Remove, RemoveAll, RemoveAt, RemoveRange, Reverse (), Reverse(Int32, Int32), Sort (), Sort(Comparison <T >) , Sort(IComparer <T >) , Sort(Int32, Int32, IComparer <T >) , ToArray, ToString, TrimExcess, TrueForAll, …

- Insert
- Inserts an element into the List <T > at the specified index.

Идем на страницу этого метода ->

List <T>.Insert Method

Inserts an element into the List <T > at the specified index.

Syntax

public void Insert(int index, T item)

Parameters

- indexType: System .Int32
  The zero-based index at which item should be inserted.
- itemType: T
  The object to insert. The value can be null for reference types.

Exceptions

ExceptionCondition

- ArgumentOutOfRangeExceptionindex is less than 0.
- -or- index is greater than Count.

Remarks

- List <T > accepts null as a valid value for reference types and allows duplicate elements.
- If CountIf Count already equals CapacityIf Count already equals Capacity, the capacity of the List <T > is increased by automatically reallocating the internal array, and the existing elements are copied to the new array before the new element is added.
- If index is equal to CountIf index is equal to Count, item is added to the end of List <T >.
- This method is an O(n) operation, where n is Count.

Examples

- The following code example demonstrates the Insert method, along with various other properties and methods of the List <T >The following code example demonstrates the Insert method, along with various other properties and methods of the List <T >

See Also

- **Reference**
- List <T > Class
- List <T > Members
- System.Collections.Generic Namespace
- InsertRange
- Add
- Remove

- И так далее!

Работа с файлами

Общие принципы работы с файлами

- ***Чтение*** *(ввод)* — передача данных с внешнего устройства в оперативную память, обратный процесс — ***запись*** *(вывод)*.
- Ввод-вывод в C# выполняется с помощью подсистемы ввода-вывода и классов библиотеки .NET. Обмен данными реализуется с помощью потоков.
- ***Поток*** (stream) — абстрактное понятие, относящееся к любому переносу данных от источника к приемнику. Потоки обеспечивают надежную работу как со стандартными, так и с определенными пользователем типами данных, а также единообразный и понятный синтаксис.
- Поток определяется как последовательность байтов и не зависит от конкретного устройства, с которым производится обмен.
- Обмен с потоком для повышения скорости передачи данных производится, как правило, через ***буфер***. Буфер выделяется для каждого открытого файла.

Классы .NET для работы с потоками

Уровни обмена с внешними устройствами

Выполнять обмен с внешними устройствами можно на уровне:

- *двоичного представления данных*
  - (BinaryReader, BinaryWriter);
- *байтов*
  - (FileStream);
- *текста*, то есть символов
  - (StreamWriter, StreamReader).

Доступ к файлам

- *Доступ* к файлам может быть:
  - ***последовательным*** - очередной элемент можно прочитать (записать) только после аналогичной операции с предыдущим элементом
  - *произвольным*, или ***прямым***, при котором выполняется чтение (запись) произвольного элемента по заданному адресу.

- Текстовые файлы - только последовательный доступ
- В двоичных и байтовых потоках можно использовать оба метода доступа

- Прямой доступ в сочетании с отсутствием преобразований обеспечивает высокую скорость получения нужной информации.

Пример чтения из текстового файла

```
static void Main()   // весь файл -> в одну строку
{       try
        {
            StreamReader f = new StreamReader( "text.txt" );
            string s = f.ReadToEnd();
            Console.WriteLine(s);
            f.Close();
        }
        catch( FileNotFoundException e )
        {
            Console.WriteLine( e.Message );
            Console.WriteLine( " Проверьте правильность имени файла!" );
            return;
        }
        catch
        {
       Console.WriteLine( " Неопознанное исключение!" );
             return;
        }
}
```

Построчное чтение текстового файла

```
StreamReader f = new StreamReader( "text.txt" );
    string s;
    long i = 0;

    while ( ( s = f.ReadLine() ) != null )
        Console.WriteLine( "{0}: {1}", ++i, s );
    f.Close();
```

Чтение целых чисел из текстового файла (вар. 1)

```
try {
        List<int> list_int = new List<int>();
        StreamReader file_in = new StreamReader( @"D:\FILES\1024" );
        Regex regex = new Regex( "[^0-9-+]+" );
        List<string> list_string = new List<string>(
                        regex.Split( file_in.ReadToEnd().TrimStart(' ') ) );
        foreach (string temp in list_string)
                list_int.Add( Convert.ToInt32(temp) );

        foreach (int temp in list_int) Console.WriteLine(temp);
        ...
}
catch (FileNotFoundException e)
        {  Console.WriteLine("Нет файла" + e.Message); return; }
catch (FormatException e)
        {  Console.WriteLine(e.Message); return;     }
catch { … }
```

Чтение чисел из текстового файла – вар. 2

```
try {
        StreamReader file_in = new StreamReader( @"D:\FILES\1024" );
        char[] delim = new char[] { ' ' };
        List<string> list_string = new List<string>(
                file_in.ReadToEnd().Split( delim,
                        StringSplitOptions.RemoveEmptyEntries ));
      List<int> list_int = list_string.ConvertAll<int>(Convert.ToInt32);
        foreach ( int temp in ist_int ) Console.WriteLine( temp );
        ...
}
catch (FileNotFoundException e)
        {  Console.WriteLine("Нет файла" + e.Message); return;  }
catch (FormatException e)
        {  Console.WriteLine(e.Message); return;   }
catch { … }
```

Работа с каталогами и файлами

Пространство имен System.IO: классы

- Directory, DirectoryInfo
- File, FileInfo

(создание, удаление, перемещение файлов и каталогов, получение их свойств)

Элементы класса DirectoryInfo

Элемент	Описание
`Create` `CreateSubDirectory`	Создать каталог или подкаталог по указанному пути в файловой системе
`Delete`	Удалить каталог со всем его содержимым
`GetDirectories`	Возвратить массив строк, представляющих все подкаталоги
`GetFiles`	Получить файлы в текущем каталоге в виде массива объектов класса FileInfo
`MoveTo`	Переместить каталог и все его содержимое на новый адрес в файловой системе
`Parent`	Возвратить родительский каталог

Пример: копирование файлов *.jpg - 1

```
class Class1                    // из каталога d:\foto в каталог d:\temp
  static void Main() {
    try {
      string destName = @"d:\temp\";
    DirectoryInfo dir = new DirectoryInfo( @"d:\foto" );
      if ( ! dir.Exists )          // проверка существования каталога
      {
      Console.WriteLine( "Каталог " + dir.Name + " не сущ." );
      return;
      }
    DirectoryInfo dest = new DirectoryInfo( destName );
    dest.Create();                    // создание целевого каталога
```

Пример: копирование файлов *.jpg - 2

```
    FileInfo[] files = dir.GetFiles( "*.jpg" ); //список файлов
    foreach( FileInfo f in files )
          f.CopyTo( dest + f.Name );        // копирование файлов

    Console.WriteLine( "Скопировано файлов " + files.Length);

  }  // конец блока try

   catch ( Exception e )
        {
            Console.WriteLine( "Error: " + e.Message );
        }
}}
```

Рубежный контроль по модулю 2

- Теор. часть (10 вопросов, 5 баллов):
  - Оператор switch -1
  - Параметры методов (передача по значению и по ссылке) – 3
  - Принципы ООП – 2
  - Синтаксис обращения к полям и методам, спецификаторы, описание объектов – 4
- Практ. часть (задача, 5 баллов):
  - Задание на массивы (на сайте) или задание по выбору преподавателя.

Практическая часть и по первому, и по второму модулю выполняется на компьютере на лабораторной работе в присутствии преподавателя (дома – только в виде исключения)

Экзамены

- 1100, 1101 – 25.01
- 1120, 1125 – 26.01
- 1105, 1106 – 29.01

- Досрочный экзамен 25.12 (старосты -> списки)