



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ИНСТИТУТ ТЕХНОЛОГИЙ (ФИЛИАЛ) ФЕДЕРАЛЬНОГО  
ГОСУДАРСТВЕННОГО БЮДЖЕТНОГО ОБРАЗОВАТЕЛЬНОГО  
УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»  
В Г. ВОЛГОДОНСКЕ РОСТОВСКОЙ ОБЛАСТИ**

**(Институт технологий (филиал) ДГТУ в г. Волгодонске)**

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ**  
**для проведения лабораторного практикума**  
по дисциплине  
«Инструментальные средства информационных систем»  
для обучающихся по направлению подготовки  
*09.03.02 Информационные системы и технологии*  
программа бакалавриата «Информационные системы»

г. Волгодонск

2021

Практическая работа №1 С - ОТЛАДКА ПРИЛОЖЕНИЙ В VISUAL STUDIO .....	3
Практическая работа №2 Инструменты создания классов в MS VS С .....	14
Практическая работа №3 Инструменты информационного поиска. Метасимволы. ....	17
Практическая работа №4. Использование инструментария операционной системы (ОС) Windows. Написание простого браузера. ....	20
Практическая работа №5. Делегаты. ....	26
Практическая работа №6. Интерфейс IEquatable.....	33
Практическая работа №7. Групповые интерфейсы. ....	37
Практическая работа №8. Списки. ....	40
Практическая работа №9. Работа с базами данных SQL. ....	47
Практическая работа №10. Регулярные выражения. ....	51
Практическая работа №11. Поиск совпадений в объектах методами IComparable.....	58

## Практическая работа №1 С - ОТЛАДКА ПРИЛОЖЕНИЙ В VISUAL STUDIO

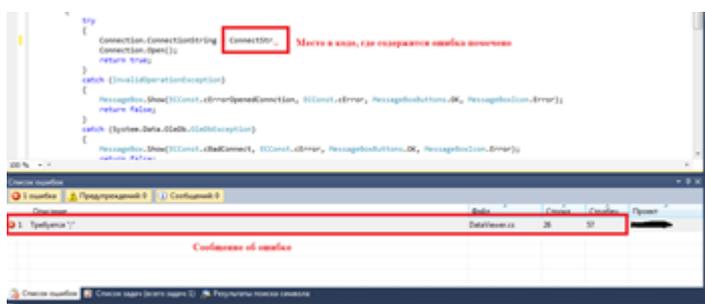
**Цель работы:** Освоить возможности стандартной системы отладки интегрированной среды разработки MS VS 2010 (15). Язык программирования C#.

**Задание:** Написать программу в технологии «WindowsForms» на языке C# для сравнения двух объектов произвольного класса. Прикладная область для реализации класса задается преподавателем (например, принтеры, мониторы, компьютеры, планшеты и т.п.).

### Теоретический материал.

Фиксация ошибок на этапе разработки.

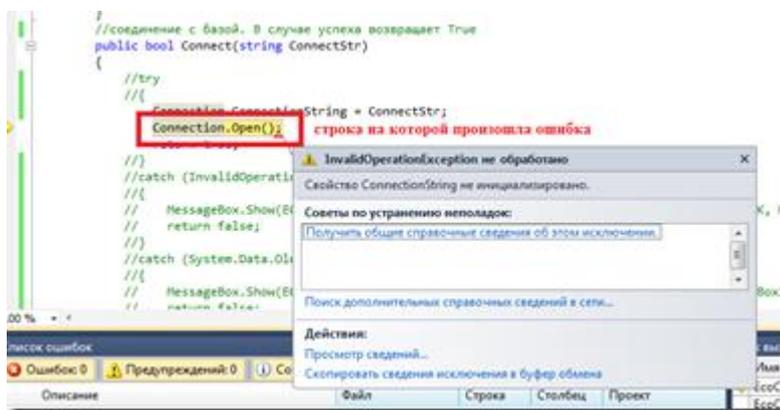
С этой функцией Visual Studio может столкнуться кто угодно, в особенности новички в том или ином языке программирования. В случае, если на этапе написания какого-либо кода программист допускает ошибку, которая может быть проанализирована ещё ДО компиляции и запуска программы – Visual Studio об этом сообщит в окне “Список ошибок”. На следующем рисунке представлен пример вывода сообщения об ошибке – в коде отсутствует точка с запятой:



В основном на этапе разработки фиксируются, если можно так выразиться, элементарные ошибки: пропуск запятой, несоответствие видимости классов, полей и т.д., то ошибки, которые могут быть исправлены быстро.

Фиксация ошибок на этапе выполнения программы

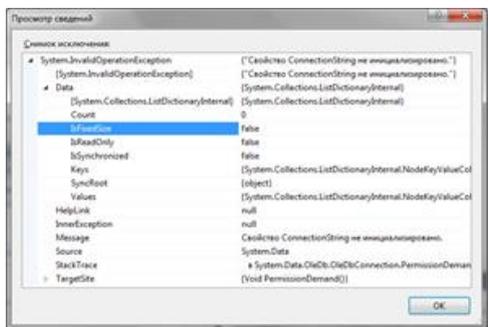
Допустим, что код у нас написан правильно с точки зрения синтаксиса, все объекты правильно определены и Visual Studio готова запустить программу на выполнение. Довольно часто могут возникать ситуации, когда на этапе выполнения программы в какой-либо метод передается неверный параметр, скажем, в качестве пути к локальной базе данных, может случайно передаться пустая строка – в этом случае обязательно возникнет исключительная ситуация. Такие моменты студия также фиксирует, если приложение запускается из среды IDE. Например, на следующем рисунке показан вывод сведений об ошибке, рассмотренной выше – я специально передал в качестве пути к локальной БД пустую строку и никак не защитил опасный код:



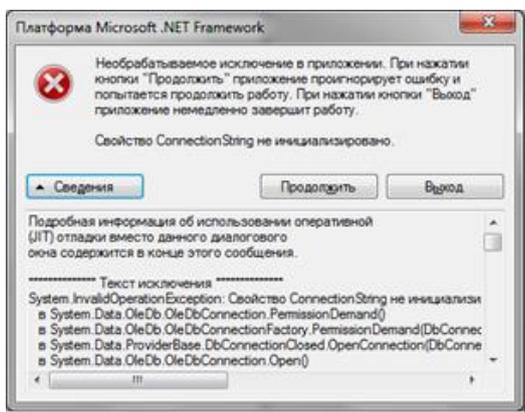
Как видим, VS выводит окно, в котором дается небольшое пояснение об исключительной ситуации. Здесь вы можете:

- просмотреть сведения об ошибке в справочных материалах.
- поискать решение проблемы в сети.
- просмотреть сведения об ошибке.
- скопировать сведения исключения в буфер обмена.

При выборе опции “Просмотр сведений” откроется ещё одно окно в котором содержится подробная информация об исключительной ситуации:



Сведения об этой исключительной ситуации при запуске программы не через VS выглядят так:



Как видим Visual Studio дает вполне исчерпывающий ответ на то, какая исключительная ситуация произошла, а также может попробовать найти возможное решение проблемы. Что если этих сведений нам окажется не достаточно? В этом случае мы приступим как раз к работе с отладчиком и будем искать проблему самостоятельно.

### Работа с отладчиком в Visual C#

Остановимся все на той же ошибке соединения с базой данных. Для решения проблемы нам необходимо узнать текущие значения переменных на момент доступа к БД. Для этого воспользуемся пошаговой отладкой – будем шаг за шагом проверять действия программы и фиксировать значения переменных.

Так как программа может состоять из тысяч и тысяч строк кода, то пройтись по каждой строке – потребуется слишком много времени, да и такой процесс отладки, в принципе, излишний – мы же как разработчики можем хотя бы примерно себе представить в каком месте кода может быть ошибка? Поэтому воспользуемся первой возможностью отладчика – установка точек останова программы.

**Точка останова или breakpoint** – это преднамеренная остановка выполнения программы, при котором выводится вывод отладчика.

Чтобы установить breakpoint необходимо установить курсор на нужной строке и выбрать в меню “Отладка – Точка останова” или нажать [F9]. При этом строка с точкой останова будет помечена:

```
//try
//{
    Connection.ConnectionString = ConnectStr;
    Connection.Open();
    return true;
//}
```

Для примера, я решил вначале остановиться на предшествующей ошибке строке. Теперь запускаем программ у из IDE, нажав [F5]. Как только выполнение программы дойдет до точки останова – выполнение прервется и запустится отладчик. При этом вид строки на которой произошла остановка будет следующим:

```
//try
//{
    Connection.ConnectionString = ConnectStr;
    Connection.Open();
    return true;
//}
```

Проверим вначале, что содержит переменная ConnectStr. Для этого подводим к ней курсор мыши и через сек унду видим следующее сообщение отладчика:

```
//try
//{
    Connection.ConnectionString = ConnectStr;
    Connection.Open();
    return true;
//}
```

ConnectStr

Во всплывающем окне мы видим, что переменная содержит пустую строку – вот и ошибка. Остается теперь и справиться её и дело сделано. приведенный выше пример поиска проблемы довольно простой, более того, так к ак это только пример – я знал в каком месте произошла ошибка и без проблем мог её найти и исправить. Одр ако в особо тяжелых случаях на отладку программы может потребоваться гораздо большее количество време ни. Для таких случаев в Visual Studio содержатся довольно удобные средства для логирования процесса отла дки.

Например, мы можем сохранить результат отладки до следующего сеанса. Для этого необходимо во всплыва ющем окне нажать кнопку справа и закрепить сообщение отладчика:

```
//try
//{
    Connection.ConnectionString = ConnectStr;
    Connection.Open();
    return true;
//}
```

ConnectStr

**закрепленное окно**

Здесь же во всплывающем окне мы можем оставить осмысленный комментарий, чтобы при следующем сеанс е отладки было легче разобраться почему мы закрепили именно этот результат, а не другой и т.д. Комментар ии выглядят следующим образом:

```
//try
//{
    Connection.ConnectionString = ConnectStr;
    Connection.Open();
    return true;
//}
//catch (InvalidOperationException)
//{
    // MessageBox.Show(EConst.cErrorOpenedConnect, EConst.cError, MessageBoxButtons.OK, MessageBoxIcon.Error);
    return false;
//}
//catch (System.Data.OleDb.OleDbException)
//{
    // MessageBox.Show(EConst.cBadConnect, EConst.cError, MessageBoxButtons.OK, MessageBoxIcon.Error);
    return false;
//}
```

ConnectStr

Оставляем комментарий, для того, чтобы не забыть :)

Свернуть комментарий

В случае более сложного процесса отладки можно сохранять не только значение отдельной переменной, но и целые деревья объектов и значения их свойств, например так:

```
//try
//{
    Connection.ConnectionString = ConnectStr;
    Connection.Open();
    return true;
//}
//catch (InvalidOperationException)
//{
    // MessageBox.Show(EConst.cErrorOpenedConnect, EConst.cError, MessageBoxButtons.OK, MessageBoxIcon.Error);
    return false;
//}
//catch (System.Data.OleDb.OleDbException)
//{
    // MessageBox.Show(EConst.cBadConnect, EConst.cError, MessageBoxButtons.OK, MessageBoxIcon.Error);
    return false;
//}
```

Connection (System.Data.Common.DbConnection) Connection

- Connection.ConnectionString (System.Data.OleDb.OleDbConnection) ""
- Connection.ConnectionTimeout (System.Data.OleDb.OleDbConnection) 15
- Connection.Database (System.Data.OleDb.OleDbConnection) ""
- Connection.DataSource (System.Data.OleDb.OleDbConnection) ""
- System.Data.OleDb.OleDbConnection, hidden (System.Data.OleDb.OleDbConnection) hidden
- Connection.InnerConnection (System.Data.ProviderBase.DbConnectionClosedNeverOpened)

Комментарий

Теперь, когда мы закрепили окно с информацией отладчика и нашли проблему, можно попробовать её исправить. Для этого останавливаем выполнение программы, нажав [Shift+F5] – окно с информацией отладчика исчезнет. Исправляем ошибку, в нашем случае достаточно передать в переменную путь к БД. Снова запускаем программу через [F5] и смотрим на вывод отладчика:



На рисунке представлен вывод по объекту connection. Так как мы находимся на строке в которой свойству ConnectionString передается значение, то мы ещё не видим решена ли проблема или нет. Мы, конечно, можем снова вызвать всплывающее окно со значением переменной ConnectStr, а можем поступить иначе – по нажатию клавиши [F11] переместиться на строку ниже и увидеть результат:



Как видно по рисунку те свойства, которые были изменены в окне отладчика выделяются красным цветом. Нажимая [F11] мы можем пошагово пройти хоть по всей программе и отследить изменения переменных.

Кстати, всю информацию по подсказкам, которые мы с Вами устанавливаем в ходе отладки можно экспортировать в отдельный XML-файл, для чего необходимо в меню выдрать “Отладка – Экспорт подсказок по данным” и потом, в случае необходимости снова импортировать в Visual Studio.

В этом посте мы рассмотрели только самый-самые основные моменты по работе с отладчиком в Visual C#. На самом деле возможности студии по отладке программ намного мощнее. Но, думаю, что в будущем мы ещё не раз вернемся к теме работы с отладчиком и рассмотрим все возможности отладки программ более подробно.

Далеко не всегда удастся писать код абсолютно без ошибок. Если ошибки компиляции нам помогает отловить компилятор, то с ошибками логики дело обстоит немного сложнее. Как узнать, почему какая-то переменная не изменяется или почему результат выполнения какого-то кода не такой, как вы ожидали? Все это сложно, если в вас нет мощного средства отладки программы, а в Visual Studio средства отладки достаточно мощные, чтобы найти любые ошибки.

Сейчас я рассмотрю примеры отладки программ на примере среды разработки Microsoft Visual C# Express Edition. Это базовая поставка среды разработки, которая доступна бесплатно. Уже в ней заложены мощные средства отладки приложений, так что в платных версиях вы сможете найти все то же самое.

Пример программы.

1. **private void** MyButton\_Click(**object** sender, EventArgs e)
2. {
3. **int** x = 10;
- 4.
5. **int** y = 15;
- 6.
7. x = Double(x);
- 8.
9. x = x / (y - 15);

```
10.
11.  MessageBox.Show(x.ToString());
12. }
```

В этом коде в третьей строке происходит вызов метода Double. Это не какой-то стандартный метод, его нужно написать в коде формы:

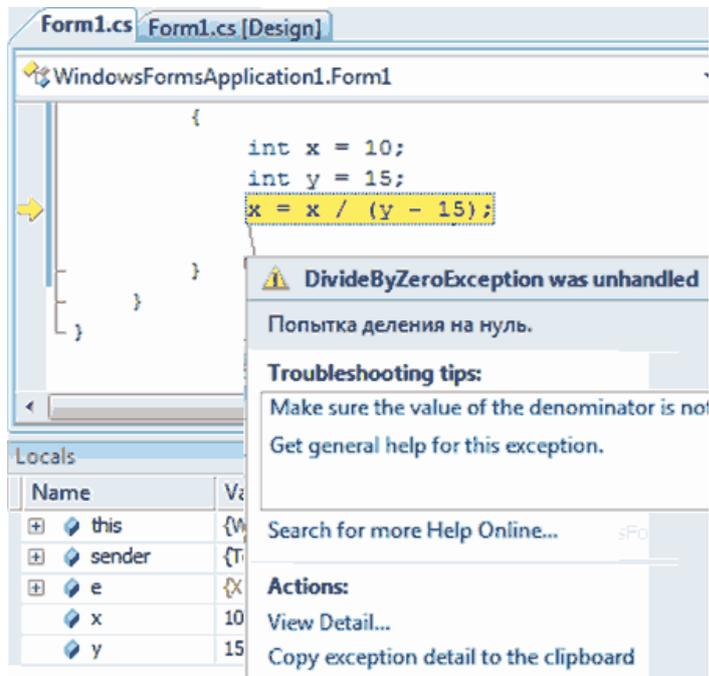
```
1.  int Double(int x)
2.  {
3.      return x * 2;
4.  }
```

Полный исходный код формы будет выглядеть следующим образом:

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.ComponentModel;
4.  using System.Data;
5.  using System.Drawing;
6.  using System.Linq;
7.  using System.Text;
8.  using System.Windows.Forms;
9.
10.
11. namespace WindowsFormsApplication1
12. {
13.     public partial class Form1 : Form
14.     {
15.         // конструктор
16.         public Form1()
17.         {
18.             InitializeComponent();
19.         }
20.
21.
22.         // обработчик события Click для кнопки
23.         private void MyButton_Click(object sender, EventArgs e)
24.         {
25.             int x = 10;
26.
27.             int y = 15;
28.
29.             x = Double(x);
30.
31.             x = x / (y - 15);
32.
33.             MessageBox.Show(x.ToString());
34.         }
35.
36.         // наш метод Double
37.         int Double(int x)
38.         {
```

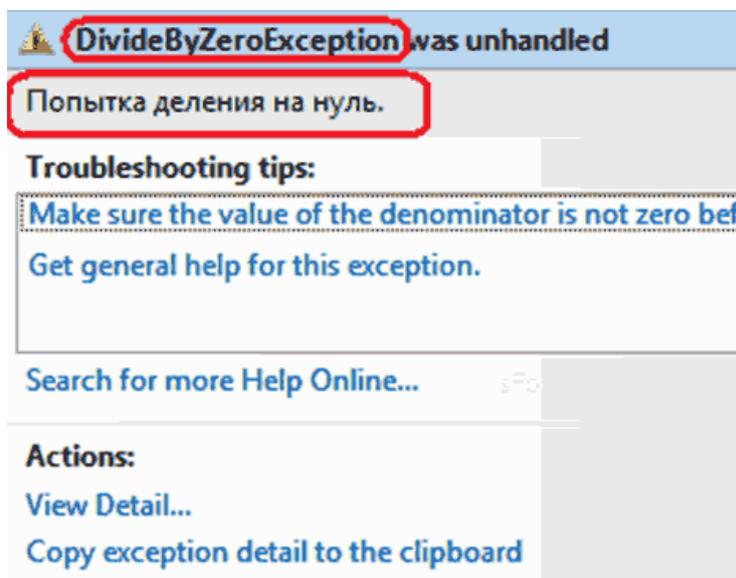
```
39.     return x * 2;
40.     }
41.     }
42. }
```

Если запустить программу из среды разработки и нажать на кнопку, то выполнение программы переключится на среду разработки и вы увидите сообщение об ошибке:



Произошла ошибка, и среда разработки взяла управление на себя, чтобы вы смогли просмотреть информацию об ошибке и попробовали определить ее источник. В редакторе кода желтым цветом выделена строка кода, в которой и произошла ошибка. От этой строки идет стрелка к всплывающему окну, в котором исключительная ситуация описана более подробно.

Название ошибки можно увидеть в заголовке всплывающего окна. Чуть ниже под заголовком находится описание ошибки. Нам повезло, потому что оно написано на русском и сразу понятно, что произошло деление на ноль:



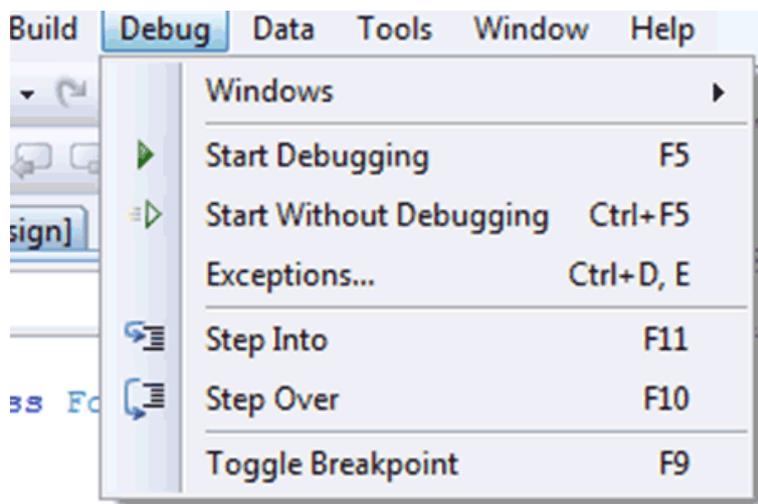
В списке Troubleshooting tips этого окна можно увидеть возможные подсказки, которые могут помочь вам найти возможное решение. В данном случае проблема проста и с ней можно разобраться без дополнительных советов, потому что в строке произошло деление на ноль и наша задача добавить перед выполнением кода проверку, которая предотвратит деление на 0. В данном случае, нужно проверить, чтобы переменная y не была равна 15:

```
1. int x = 10;
2.
3. int y = 15;
4.
5. if (y == 15)
6.     MessageBox.Show("Переменная y не должна быть равна 15 ");
7.
8. x = x / (y - 15);
9.
10. MessageBox.Show(x.ToString());
```

Но определение ошибки далеко не всегда так легко определимо. Бывают случаи, когда нужно пройти программу пошагово, строчка за строчкой, в поисках уязвимого места, которое привело к проблеме. Давайте попробуем пройти программу пошагово. Проходить всю программу шаг за шагом проблематично, потому что очень многое происходит за кулисами, в том числе и обработка событий, а большой проект проходить шаг за шагом вообще подобно самоубийству. Поэтому чаще всего заранее определяется блок кода, который нужно проанализировать. В нашем случае этим блоком кода будет обработчик события Click для кнопки.

### Конфигурация компиляции

Для того, чтобы отладка программы была доступна, программа должна быть еще запущена в режиме отладки и из среды разработки. Простое нажатие клавиши F5 как раз запускает программу в отладочном режиме. Чтобы запустить программу без возможности отладки, нужно нажать Ctrl+F5. Соответствующие пункты меню можно найти в меню Debug:



- **Debug | Start Debugging** запустить в режиме отладки
- **Debug | Start Without Debugging** запустить в режиме выполнения и без возможности отладки

Тут нужно заметить, что для отладки программа должна быть скомпилирована в конфигурации Debug, которая включает в исполняемый файл дополнительную информацию, необходимую при отладке программы. Конфигурацию можно выбрать на панели инструментов чуть правее кнопки запуска программы. На следующем снимке соответствующий выпадающий список окрашен красным цветом:



В версии Microsoft Visual C# Express Edition данный выпадающий список недоступен, а отладочная информация попадает в исполняемый файл. В версии Visual Studio Standard и более полных вариантах данный выпадающий список будет содержать конфигурации Debug и Release. Вы должны будете выбрать первый пункт. Вы можете создавать и свои конфигурации, но это уже тема отдельного разговора.

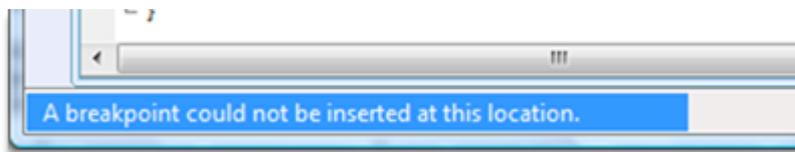
### Точки останова

Итак, нам нужно отладить метод и мы сделаем это с самого начала. Для этого нужно поставить точку прерывания в нужном нам месте. Точка прерывания - это точка в коде программы, при достижении которой выполнение программы будет прервано и управление будет передано среде разработки. Вы можете создавать точки останова в любом месте программы, но только там, где есть код и программа может прервать выполнение. Для создания точки перейдите на нужную строку и:

- нажмите **F9**
- выберите меню **Debug** | **Toggle Breakpoint**
- дважды щелкните на полоске серого цвета, слева от строки текста в окне редактора кода

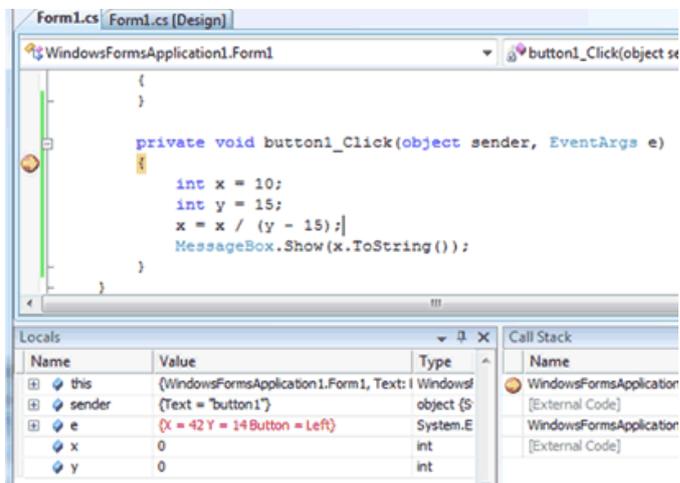
Напротив строки на полоске серого цвета слева от текста появится красный кружок, символизирующий, что здесь стоит точка останова. Если еще раз попытаться поставить точку останова на этой же строке, то точка останова будет снята.

Если точку на текущей строке установить нельзя, то в строке состояния среды разработки появится соответствующее сообщение:



Так как мы отлаживаем метод с самого начала, то поставьте курсор на строку с именем метода и нажмите F9. Теперь можно запускать программу. Кстати, запустить программу можно было и раньше, потому что точку останова можно ставить в любой момент, даже во время выполнения программы.

Запустите программу и нажмите на кнопку. После нажатия кнопки должен сработать метод `button1_Click` (у вас он может называться по-другому), на который я поставил обработчик события. Так как мы поставили точку останова, среда разработки перехватит на себя выполнение и выделит оператор, который можно выполнить следующим шагом желтым цветом (назовем эту точку курсором пошагового выполнения программы):



Обратите внимание, что выделена строка с символом открывающейся фигурной скобки. Это потому, что выполнение метода начинается именно с фигурной скобки, а не с имени, где мы поставили точку останова. Да, точка останова иногда может смещаться, но если вы поставите ее на конкретный оператор, который может быть выполнен, то точка останется там и никуда не поедет.

Внизу окна среды разработки появилось две панели:

- **Locals** - где в виде списка представлены все переменные, доступные в текущем методе. В этом списке переменные представлены в виде трех колонок:
  - Name - название переменной
  - Value - значение переменной
  - Type - тип
- **Call stack** - стек вызовов. В этом окне перечислены методы, которые уже были вызваны ранее.

### Пошаговое выполнение

Итак, наш курсор пошагового выполнения остановился на какой-то точке и теперь хотим начать тестирование кода. Посмотрим на панель, где появилась новая панелька отладки, там же мы и найдем необходимые команды (в скобках указаны горячие клавиши):



На этой панели можно найти следующие интересные кнопочки:

- **Continue (F5)** - продолжить выполнение программы.
- **Stop debugging (Shift+F5)** - остановить отладку. При этом остановится и выполнение программы. Даже больше - выполнение программы прервется на той точке, на которой сейчас и находится, т.е. оно не будет завершено корректно и ничего не сохранится, если вы в программе что-то делали;
- **Restart (Ctrl+Shift+F5)** - перезапустить программу. Выполнение программы будет прервано и запустится заново;
- **Show Next Statement (Alt + Num \*)** - показать следующий оператор, т.е. переместить курсор редактора кода в курсор пошагового выполнения. Курсор переместится в начала оператора, который должен быть выполнен следующим. В редакторе кода он выделен желтым цветом;
- **Step Into (F11)** - выполнить очередной оператор. Если это метод, то перейти в начало этого метода, чтобы начать отладку. Например, если вы находитесь на строке: `x = Double(x)` то, курсор пошагового выполнения перейдет на начало метода `Double` и вы сможете отладить этот метод;
- **Step Over (F10)** - выполнить очередной оператор. Если это метод, то он будет полностью выполнен, т.е. курсор выполнения не будет входить внутрь метода;

- **Step out (Shift + F11)** - выйти из метода. Если вы отлаживаете метод и нажмете эту кнопку, то метод выполнится до конца и курсор пошагового выполнения выйдет из метода и остановится на следующей строке после вызова данного метода. Например, если вы отлаживаете метод `Double`, нашего примера и нажмете эту кнопку, то метод выполнится до конца, а выполнение остановится на строке `"x = x / (y - 15);"` метода `button1_Click`.

Попробуйте сейчас пошагово выполнить код метода, нажимая клавишу **F10**. Потому запустите снова приложение и попробуйте пошагово выполнить его, нажимая клавишу **F11**. Когда произойдет ошибка, попробуйте прервать работу отладки и приложения, нажав **Shift+F5**.

## Просмотр значений

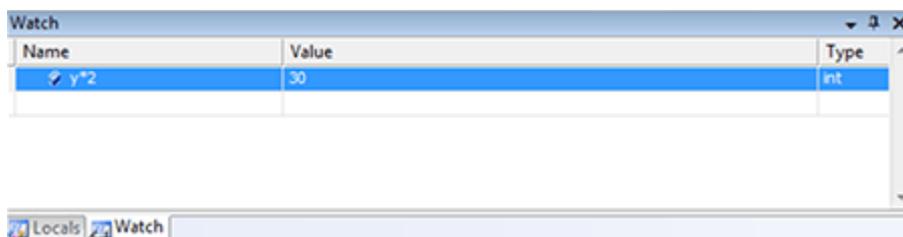
Выполните пошагово код так, чтобы курсор выполнения остановился на следующей строке: `x = x / (y - 15);`

Нам нужно узнать, почему происходит ошибка. Так как ошибка в делении, то нужно посмотреть, на что происходит деление. В данном случае это скобка. Наведите мышкой на открывающуюся или закрывающуюся скобку и вы увидите всплывающую подсказку, в которой находится результат вычисления значения в скобках:

```
private void button1_Click(object :
{
    int x = 10;
    int y = 15;
    x = Double(x);
    x = x / (y - 15);
    MessageBd (y - 15)|0 ToString();
}
```

Результат находится после вертикальной черты и он равен нулю. Вы можете наводить мышкой на любую переменную, и отладчик покажет вам ее значение в виде всплывающей подсказки.

Посмотрите на панель `Locals`. По идее внизу у вас должны быть еще одна закладка `Watch`. Переключитесь на нее. Здесь находится список, в который вы можете вносить свои переменные и выражения, за значениями которых вы хотите наблюдать. Выделите первую пустую строку в списке (всегда одна пустая строка присутствует в списке) и в поле `Name` введите `y*2` и нажмите `Enter`. В поле `Value` появится значение выражения. После каждого шага выполнения значение переменной будет пересчитано и отображено. Таким образом, вам не нужно после каждого шага наводить на переменную, чтобы узнать ее значение. Оно видно в окне `Watch`:



На закладке `Locals` видны переменные, которые актуальны для данного метода. Сюда включаются переменные, объявленные внутри метода и параметры метода, а так же переменная `this`. Параметры представлены в виде дерева для объектных типов данных. Раскрывая дерево объекта `this` вы можете увидеть значения всех свойств и даже объектов на текущей форме, ведь `this` всегда указывает на текущий объект, которым является форма:

Locals	
Name	Value
[-] this	{WindowsFormsApplication1.Form1, Text: Form1}
[-] base	{WindowsFormsApplication1.Form1, Text: Form1}
[-] button1	{Text = "button1"}
[-] base	{Text = "button1"}
[-] AutoSizeMode	GrowOnly
[-] DialogResult	None
[-] Non-Public members	
[-] components	null
[-] sender	{Text = "button1"}
[-] e	{X = 44 Y = 6 Button = Left}
[-] x	20
[-] y	15

## Практическая работа №2 Инструменты создания классов в MS VS C#

**Цель работы:** Изучить создание классов на языке C#. Ознакомиться с основными свойствами классов, изучить структуру их построения.

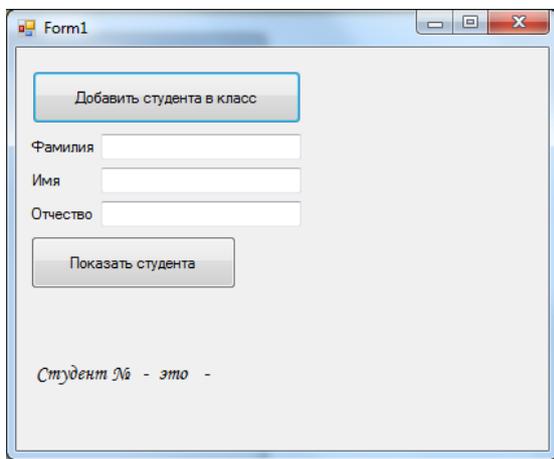
**Задание:** Создать класс по произвольной тематике (задается индивидуально преподавателем). На его основе создать структуру, по которой к нему можно будет обращаться. Добавить метод, который будет возвращать что либо, и метод, не возвращающий значения.

### Ход работы:

1. Создаем пустой проект с шаблоном Windows Forms.
2. Создаем класс под названием student. Присвоим ему значения, создадим два метода.

```
public class student//название класса
{
    public string fam_name;//фамилия
    public string name;//имя
    public string o_name;//отчество
    public string fullname;//строка для метода
    public int num;//номер для метода
    public void Plus();//метод, увеличивающий номер на 1
    {
        num++;
    }
    public string Getfullname();//метод, возвращающий полное имя
    {
        fullname = fam_name + " " + name + " " + o_name;
        return fullname;
    }
}
```

3. Далее создаем интерфейс программы. Разместим на форме поля ввода значений, поле вывода результата, текстовые сообщения и исполняющую кнопку. Отметим, что данная программа направлена исключительно на ознакомление со структурами классов. Сохраняем получившийся результат.



4. Мы получили интерфейс программы. Однако для его работоспособности необходимо прописать код.

Просматриваем получившийся код программы:

```
namespace John_project__4
```

```
{
```

```
public partial class Form1 : Form
```

```
{
```

```
public class student
```

```
{
```

```
public string fam_name;
```

```
public string name;
```

```
public string o_name;
```

```
public string fullname;
```

```
public int num;
```

```
public void Plus()
```

```
{
```

```
num++;
```

```
}
```

```
public string Getfullname()
```

```
{
```

```
fullname = fam_name + " " + name + " " + o_name;
```

```
return fullname;
```

```
}
```

```
}
```

```
student a = new student();
```

```
public Form1()
```

```
{  
    InitializeComponent();  
}
```

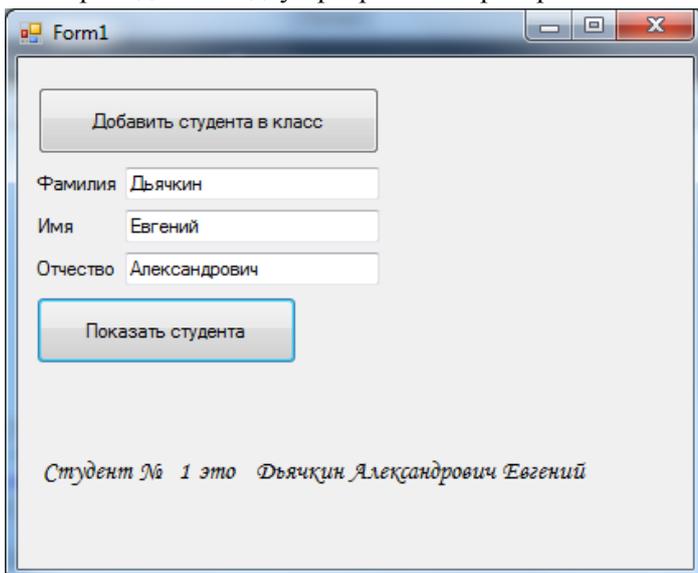
```
private void button1_Click(object sender, EventArgs e)
```

```
{  
    a.Plus();//присвоение номера в классе  
    a.fam_name = textBox1.Text;  
    a.name = textBox2.Text;  
    a.o_name = textBox3.Text;  
}
```

```
private void button2_Click(object sender, EventArgs e)
```

```
{  
    label3.Text = Convert.ToString(a.num);  
    label4.Text = a.Getfullname();  
}  
}  
}
```

5. Проводим отладку программы. Проверим выполнение класса.



## Практическая работа №3 Инструменты информационного поиска. Метасимволы.

**Цель работы:** Провести выборку ссылок из произвольного Web ресурса.

**Задание:** URL адрес ресурса задается преподавателем для каждого студента индивидуально.

### Теоретический материал.

Т а б л и ц а 1 – Метасимволы в регулярных выражениях

Символ	Значение
<code> 0</code>	Символ <i>NULL</i>
<code> t</code>	Символ табуляции
<code> v</code>	Символ вертикальной табуляции
<code> r</code>	Символ возврата каретки.
<code> n</code>	Символ перевода строки.
<code> f</code>	Символ перевода формата.
<code> b</code>	Граница слова, т. е. позиция между словом и пробелом или переводом строки.
<code> B.</code>	Любая позиция кроме границы слова
<code> xhh</code>	Символ с кодом <i>hh</i> (2 шестнадцатиричных цифры).
<code>[abc]</code>	Любой из перечисленных в скобках символов. Используя тире можно указать диапазон символов, например, <code>[a-f]</code> – то же самое, что <code>[abcdef]</code> .
<code>[^abc]</code>	Любой символ, кроме перечисленных в скобках.
<code> d</code>	Цифра. Эквивалентно <code>[0-9]</code> .
<code> D</code>	Любой символ, кроме цифр. Эквивалентно <code>[^0-9]</code> .
<code> w</code>	Цифра, буква (латинский алфавит) или знак подчеркивания. Эквивалентно <code>[0-9a-zA-Z_]</code> .
<code> W</code>	Любой символ, кроме цифр, букв (латинский алфавит) и знака подчеркивания. Эквивалентно <code>[^0-9a-zA-Z_]</code> .
<code> s</code>	Пробельный символ (пробел, табуляция, перевод строки и т. п.).
<code> S</code>	Любой символ, кроме пробельных.
<code>.</code>	Любой символ, кроме перевода строки.
<code> s S </code>	Для поиска любого символа, включая перевод строки,
<code> </code>	Экранирование. Например, символ «.» является спецсимволом, если же нужно чтобы он воспринимался буквально, т. е. означал именно точку, то нужно записать так: «\.»
<code>a b</code>	Условие «ИЛИ» ( <i>a</i> или <i>b</i> ). Так как в данном случае с обеих сторон разделителя ровно по одному символу, то можно заменить на <code>[ab]</code>
<code>(abc)</code>	Подвыражение. Соответствующий подвыражению текст запоминается отдельно от общего результата
<code>(?:abc)</code>	Подвыражение. Соответствующий подвыражению текст не запоминается. Используется для группировки частей образца, например, <code>/ко(?:т шка)/</code> то же самое, что <code>/кот кошка/</code>
<code>(?=abc)</code>	Поиск с «заглядыванием вперед», срабатывает, если соответствие найдено. Например, <code>/Windows (?=95 98 NT 2000)/</code> найдет «Windows » в строке «Windows 98», но ничего не найдет в строке «Windows 3.1». После сопоставления поиск продолжается с позиции, следующей за найденным соответствием, без учета заглядывания вперед
<code>(?!abc)</code>	Поиск с «заглядыванием вперед», срабатывает если соответствие <b>не найдено</b> . Например, <code>/Windows (?=95 98 NT 2000)/</code> найдет «Windows » в строке «Windows 3.1», но ничего не найдет в строке «Windows 98». После сопоставления поиск продолжается с позиции, следующей за найденным соответствием, без учета заглядывания вперед

Также существуют метасимволы, которые не только описывают нужные символы или управляющие последовательности, но и положение исходных символов, а также количество повторов.

Они приведены в таблице 2.

Таблица 2 – Метасимволы, уточняющие местоположение символов

Символ	Значение
^	Начало входного текста (^В – В, но только как первый символ текста)
\$	Конец входного текста (X\$ – X, но только как последний символ текста)
*	Предыдущий символ может повторяться 0 или более раз (on*e – oe, one, onne, onnne и т.д.)
+	Предыдущий символ может повторяться 1 или более раз (on+e – one, onne, onnne ... (но не oe))
?	Предыдущий символ может повторяться 0 или 1 раз (on?e – oe, one)
\f	Символ перевода формата.
\b	Граница слова, т. е. позиция между словом и пробелом или переводом строки.

Используя различные комбинации из метасимволов и символов, можно создавать шаблоны, которые будут найдены в тексте.

## Пример программы.

### 1. Подключаем дополнительные библиотеки

```
using System.IO;
using System.Web;
using System.Net;
using System.Text.RegularExpressions;
```

### Код программы:

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            string a = "http://www.vis2.ru/";
            string html = string.Empty;
            HttpRequest request1 = (HttpRequest)HttpRequest.Create(a);
            HttpResponse response1 = (HttpResponse)request1.GetResponse();
            StreamReader st1 = new StreamReader(response1.GetResponseStream(), Encoding.UTF8);
            html = st1.ReadToEnd();
            response1.Close();
            Regex Red2 = new Regex(@" src[^\s];]+ ");
            Match cool2 = Red2.Match(html);
            while (cool2.Success)
            {
                Console.WriteLine(cool2.Value);
                cool2 = cool2.NextMatch();
            }
            //Console.WriteLine(html);
            Console.Read();
        }
    }
}
```



## Практическая работа №4. Использование инструментария операционной системы (ОС) Windows. Написание простого браузера.

**Цель работы:** Создать интерфейс и написать программу на C# для работы браузера.

**Здание:** Реализовать в браузере переходы «Вперед», «Назад», «Домашняя страница», «Переход по заданному адресу URL»

### Теоретический материал.

Создаём свой веб браузер с помощью Windows Forms C#

Создаваемый браузер будет работать на движке IE, поэтому про его быстродействие можно и не думать.

Итак заходим в VS C# и создаем новый проект Windows Forms.

Появившуюся форму можно растягивать как хотим, что мы и сделаем дабы увеличить наше окошко.

1.Далее из ToolBox выбираем и перетаскиваем на форму элемент управления WebBrowser.

Если у вас WebBrowser занял всё место на форме то жмём на треугольнике в правом верхнем углу левой кнопкой мыши и выбираем -> Unlock in Parent Container.

2.Следующий шаг это перетащить с тулбокса(ToolBox) на форму текстовый (textBox) (в нее будем вводить адрес URL).

3.Поместим её в верхней части экрана. Далее кликаем правой кнопкой мыши на текстовке и вызываем свойства (properties) находим свойство Dock и устанавливаем его значение в Top.

4.Далее вспоминаем как мы делали шаг №1, и кликаем на треугольнике на элементе управления веб браузер (WebBrowser) и жмём Dock in Parent Container.

По умолчанию Visual Studio дало текст боксу и веб браузеру имена textBox1 и webBrowser1, хорошим стилем программирования считается менять такие имена на более понятные, то есть имя textBox1 не несёт в себе никакого смысла, вот если так textBoxUrl, то любой человек догадается, что этот текстовый предназначен для введения веб адреса.Также свойство Text элемента управления textBoxUrl я изменил с значения пустой строки на фразу Enter web adress,для того чтобы даже самый несмышлёный пользователь понял что в это поле нужно ввести название веб адреса.

Имя webBrowser1 изменено на webBrowser, так как у нас будет использоваться только один элемент управления то индекс 1 можно и убрать.

Пока что наш браузер будет выполнять только одну функцию: после того как пользователь ввёл веб адрес и нажал на клавишу "Enter" браузер загрузит выбранную пользователем страницу.

5.Заходим в свойства текст бокса и переключаемся в раздел Events (события)

6.Находим в списке событий KeyDown и кликаем в поле напротив него два раза.

Получили сгенерированный студией код:

Вот этот код:

```
private void textBoxUrl_KeyDown(object sender, KeyEventArgs e)
{
}
```

И так мы знаем что в C# базовым для всех классов выступает класс object, и собственно наш параметр sender представляет объект представляющий событие, а второй параметр e который имеет тип KeyEventArgs описывает это событие, то есть при помощи параметра e мы будем узнавать какая клавиша нажата когда выделен элемент текстовый.

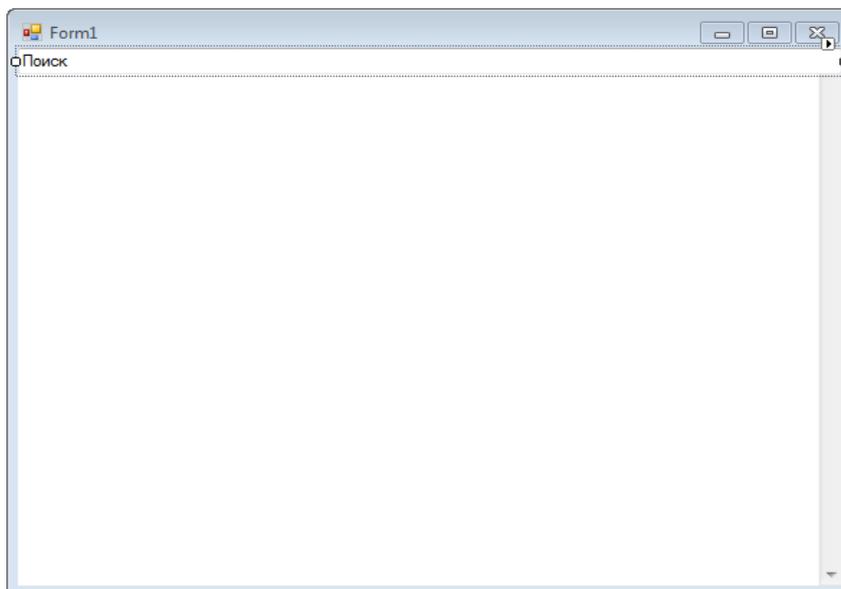
теперь в данный блок прописываем такой код:

```
1. if (e.KeyCode.ToString() == "Return") //если
нажатая клавиша enter
```

2. `webBrowser.Navigate(@"http://www." + textBoxUrl.Text);` /\* //то предаём методу Navigate нашего элемента управления webBrowser
  3. //метод Navigate принимает строку - адрес веб сайта. также для того что бы постоянно не писать http://www. мы добавляем
  4. // его к каждому адресу введённому в текст бокс символ @ перед строкой означает что в этой строке не будут находиться escape
  5. //последовательности, а это значит что "/" , воспринимается как двойной слеш, а не как одинарный(см. scape последовательности в C#) \*/
- /\* This source code was highlighted with Source Code Highlighter.
7. Теперь следует протестировать браузер.

### Пример программы.

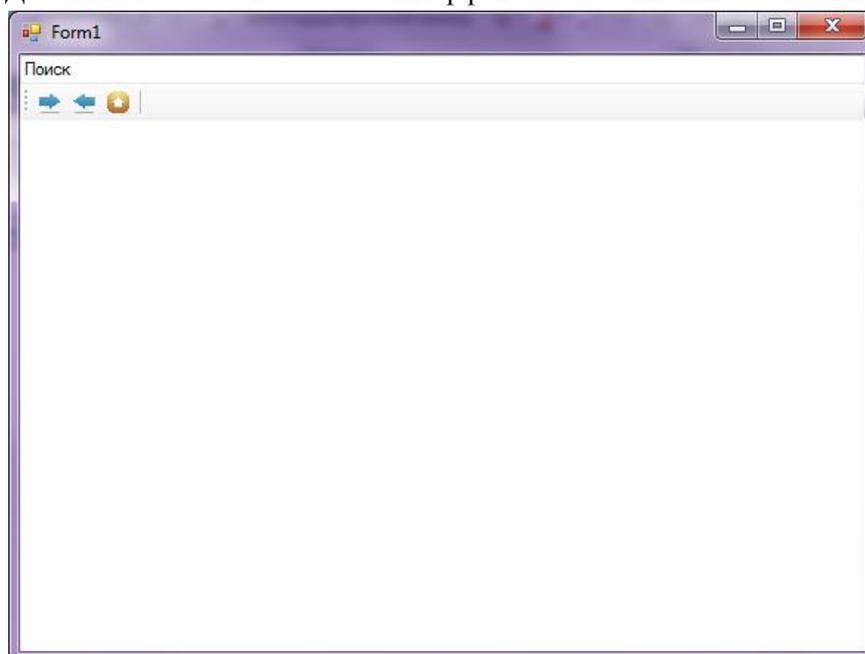
1. Создаем форму и добавляем в нее текстовое поле и функции (кнопки): вперед, назад, домой.
2. Начальная страница браузера

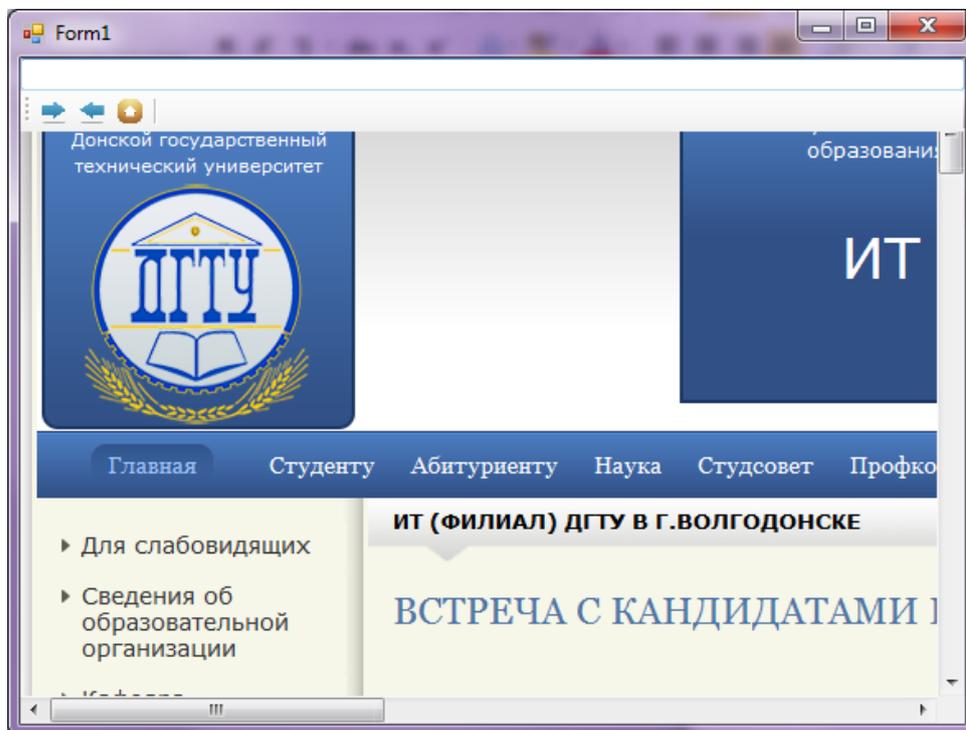


Задаем в поле адреса URL нашего института:

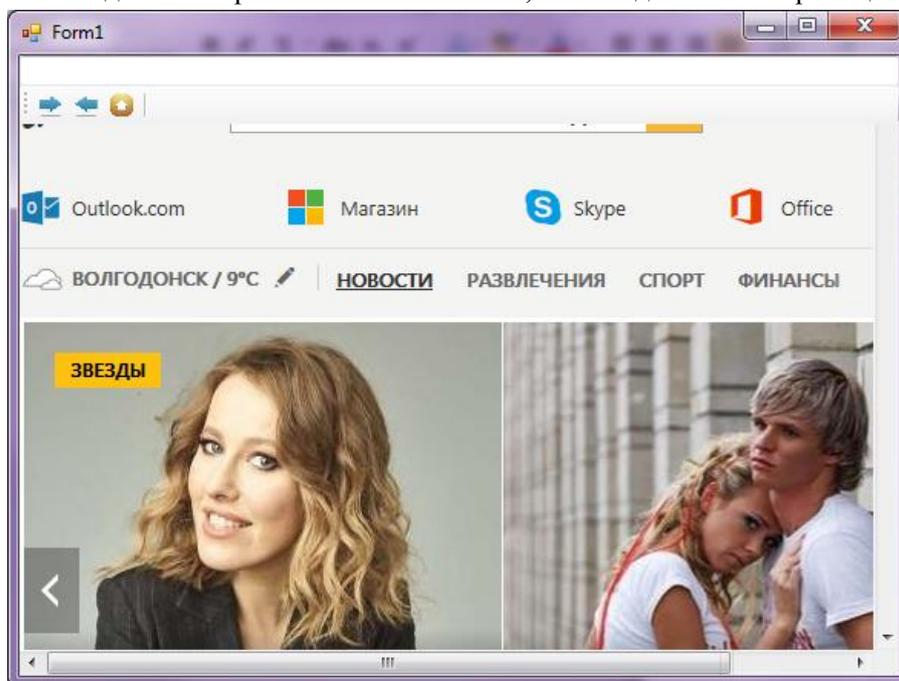


3. Добавляем на интерфейс кнопки вперед, назад, домой





4. Кнопка домой отправляет нас на новости, т.е. это домашняя страница



#### Код программы:

```
namespace lab3
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void textbox1_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode.ToString() == "Return") ;//если нажатая клавиша enter
            webBrowser.Navigate(@"http://www." + textbox1.Text);
        }
    }
}
```

```

}

private void создатьToolStripButton_Click(object sender, EventArgs e)

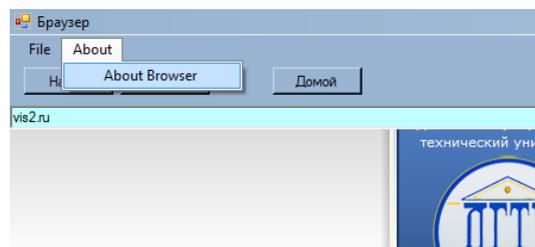
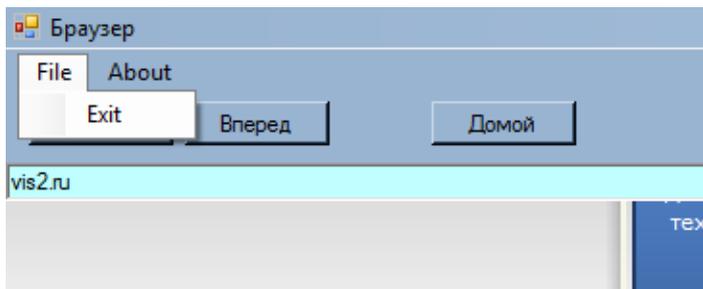
{
    webBrowser.GoForward();
}

private void Form1_Load(object sender, EventArgs e)
{
}

private void сохранитьToolStripButton_Click(object sender, EventArgs e)
{
    webBrowser.GoHome();
}

private void открытьToolStripButton_Click(object sender, EventArgs e)
{
    webBrowser.GoBack();
}
}
}

```





**Вывод:** Создали интерфейс и написали программу для работы браузера на C#.

## Практическая работа №5. Делегаты.

**Цель работы:** Рассмотреть реализовать работу в программе на языке C# с делегатами как объектами ссылающимися на методы.

**Задание:** Создать делегата и методы для переменных типа `int` и `double`, сконструировать делегат и передавать ему методы обычным способом. Методы и объекты создавать для прикладной области, указанной индивидуально преподавателем.

### Теоретический материал.

Делегат представляет собой объект, который может ссылаться на метод. Объект делегата поддерживает:

- адрес метода, на котором он вызывается;
- аргументы этого метода;
- возвращаемое значение (если есть) этого метода.

Тип делегата объявляется с помощью ключевого слова `delegate`.

`delegate возвращаемый_тип имя (список_параметров);`

где `возвращаемый_тип` обозначает тип значения, возвращаемого методами, которые будут вызываться делегатом;

`имя` – конкретное имя делегата;

`список_параметров` — параметры, необходимые для методов, вызываемых делегатом.

Делегат создается в пространстве имен, а методы – в `class Program`.

`namespace имя_проекта`

```
{  
    //создаем делегата, например для методов для переменных int  
    delegate int имяДелегата(int x, int y);
```

```
class Program
```

```
{  
    //создаем несколько методов  
    static int имя метода (int x, int y) {  
        return значение_операции_над_x,y;  
    }  
}
```

Как только делегат создан и снабжен необходимой информацией, он может динамически вызывать методы, на которые указывает, во время выполнения. В тот момент, когда происходит обращение к экземпляру делегата, вызывается метод, на который он ссылается. Следовательно, вызов метода разрешается во время выполнения, а не в процессе компиляции.

Возможно как конструировать делегаты и передавать им методы, изменяя ссылку на метод:

```
IntOperation oper = new IntOperation(Proizv);  
  
//и создаем переменную, в которую положим полученное значение  
int res = oper(2,3);  
Console.WriteLine(res);  
  
//изменяем ссылку на другой метод  
  
oper = new IntOperation(Delenie);
```

так и использовать групповое преобразование методов, позволяющее присвоить имя метода делегату, не прибегая к оператору new или явному вызову конструктора делегата:

```
//конструируем делегат
IntOperation op = new IntOperation(Func);
//используем групповое преобразование
op = Func;
//используем метод для получения значения
int result = op(10, 3);
Console.WriteLine(result);

//а теперь другой метод
op = Delenie;
result = op(36, 6);
```

При работе с делегатами можно использовать не только статические методы. Делегат может ссылаться и на методы экземпляра, хотя для этого требуется ссылка на объект. Например, в пространстве имен проекта создан делегат, и некий класс MyClass, который содержит свои методы:

```
namespace ConsoleApplication1
{
    // Создадим делегат
    delegate int IntOperation (int i, int y);
    // Создадим класс с методами
    public class MyClass
    {
        public static int Sum(int x, int y)
        {
            return x + y;
        }
    }
}
```

тогда вызов метода данного класса будет происходить аналогично приведенному выше, но с обращением к классу, в котором содержится данный метод:

```
op1 = MyClass.Sum;
```

## Пример программы.

### 1. Листинг программы:

```
using System;

namespace ConsoleApplication1
{
    // Создадим делегат
    delegate void IntOperation(int i, int j);

    // Создадим класс с методами
```

```
public class MyClass
{
    public static void Prz(int x, int y) //произведение
    {
        int p = x * y;
        Console.WriteLine("Произведение: " + Convert.ToString(p));
    }
}
```

```
class Program
{
    // Организуем ряд методов
    static void Summa(int x, int y)
    {
        int s = x + y;
        Console.WriteLine("Сумма: " + Convert.ToString(s));
    }

    static void Raznost(int x, int y)
    {
        int r = x - y;
        Console.WriteLine("Разность: " + Convert.ToString(r));
    }

    static void Delenie(int x, int y)
    {
        int d;
        if (y != 0)
        {
            d = (x / y);
            Console.WriteLine("Частное: " + Convert.ToString(d));
        }
    }
}
```

```

    }

    else

        Console.WriteLine("Делить на 0 нельзя!");

    }

static void Main()

{

    // Структурируем делегаты

    IntOperation Op;

    IntOperation Sum = new IntOperation(Summa);

    IntOperation Rzn = new IntOperation(Raznost);

    IntOperation Prz = MyClass.Prz;

    IntOperation Del = new IntOperation(Delenie);

    //Задание 1

    IntOperation op1 = new IntOperation(Summa);

    Console.WriteLine("Задание 1");

    op1(10, 5);

    // переопределим

    op1 = new IntOperation(Raznost);

    op1(10, 5);

    op1 = new IntOperation(Delenie);

    op1(10, 5);

    //Задание 2 (от класса)

    op1 = MyClass.Prz;

    Console.WriteLine(" ");

    Console.WriteLine("Задание 2 (от класса)");

    op1(10, 5);

```

//Задание 3 (свитч кейс)

```
Console.WriteLine(" ");  
Console.WriteLine("Задание 3 (свитч кейс)");  
Console.WriteLine("Введите действие (+, -, *, /)");  
string myOperat = Console.ReadLine();
```

```
sw1(myOperat);
```

//Задание 4 (Групповая адресация)

```
Console.WriteLine(" ");  
Console.WriteLine("Задание 4 (Групповая адресация)");
```

```
Op = Sum;  
Op += Rzn;  
Op += Del;  
Op += Prz;
```

// Выполняем делегат

```
Op(10, 5);
```

//Задание 5 (Групповая адресация - убираем Произведение)

```
Console.WriteLine(" ");  
Console.WriteLine("Задание 5 (Групповая адресация - убираем Произведение)");
```

```
Op -= Prz;
```

// Выполняем делегат

```
Op(10, 5);
```

```
Console.ReadLine();
```

```
}
```

```
// Данный метод выводит выбор пользователя
```

```
static void sw1(string s)
{
    switch (s)
    {
        case "+":
            Console.WriteLine("Вы выбрали сложение");
            IntOperation op1 = new IntOperation(Summa);
            Console.WriteLine("Задание 1");
            op1(10, 5);
            break;

        case "-":
            Console.WriteLine("Вы выбрали вычитание");
            op1 = new IntOperation(Raznost);
            op1(10, 5);
            break;

        case "*":
            Console.WriteLine("Вы выбрали умножение");
            op1 = MyClass.Prz;
            op1(10, 5);
            break;

        case "/":
            Console.WriteLine("Вы выбрали деление");
            op1 = new IntOperation(Delenie);
            op1(10, 5);
            break;
    }
}
```

default:

```
Console.WriteLine("Неверная операция");
```

```
break;
```

```
}
```

```
}
```

```
}
```

```
}
```

2. Окно готовой программы:

```
file://Service/free/Мешков/ИС31В ТП/ГОТОВЫЕ РАБОТЫ/Скобелева Р.А/Проекты/lab3 (delega...
Задание 1
Сумма: 15
Разность: 5
Частное: 2

Задание 2 <от класса>
Произведение: 50

Задание 3 <свитч кейс>
Введите действие (<+, -, *, />)
*
Вы выбрали умножение
Произведение: 50

Задание 4 <Групповая адресация>
Сумма: 15
Разность: 5
Частное: 2
Произведение: 50

Задание 5 <Групповая адресация - убираем Произведение>
Сумма: 15
Разность: 5
Частное: 2
```

## Практическая работа №6. Интерфейс IEquatable.

Цель работы: Получить навыки в программировании интерфейсов IEquatable для сравнения объектов.

Задание: Написать программу на языке C# для реализации интерфейса IEquatable. Методы и объекты создавать для прикладной области, указанной индивидуально преподавателем.

Теоретический материал.

### Интерфейсы (Interface)

Для начала ознакомимся с формальным определением типа интерфейса. Интерфейс (interface) представляет собой не более чем просто именованный набор абстрактных членов. Абстрактные методы являются чистым протоколом, поскольку не имеют никакой стандартной реализации. Конкретные члены, определяемые интерфейсом, зависят от того, какое поведение моделируется с его помощью. Это действительно так. Интерфейс выражает поведение, которое данный класс или структура может избрать для поддержки. Более того, каждый класс (или структура) может поддерживать столько интерфейсов, сколько необходимо, и, следовательно, тем самым поддерживать множество поведений.

Например, реализация интерфейсов IEnumerable и IEnumerator дает возможность просматривать содержимое объекта с помощью оператора foreach.

В библиотеках базовых классов .NET поставляются сотни предопределенных типов интерфейсов, которые реализуются в различных классах и структурах. Например, в состав ADO.NET входит множество поставщиков данных, которые позволяют взаимодействовать с определенной системой управления базами данных. Это означает, что в ADO.NET на выбор доступно множество объектов соединения (SqlConnection, OracleConnection, OdbcConnection и т.д.).

В интерфейсе ни у одного из методов не должно быть тела. Это означает, что в интерфейсе вообще не предоставляется никакой реализации. В нем указывается только, что именно следует делать, но не как это делать. Как только интерфейс будет определен, он может быть реализован в любом количестве классов. Кроме того, в одном классе может быть реализовано любое количество интерфейсов.

Для реализации интерфейса в классе должны быть предоставлены тела (т.е. конкретные реализации) методов, описанных в этом интерфейсе. Каждому классу предоставляется полная свобода для определения деталей своей собственной реализации интерфейса. Следовательно, один и тот же интерфейс может быть реализован в двух классах по-разному. Тем не менее в каждом из них должен поддерживаться один и тот же набор методов данного интерфейса. А в том коде, где известен такой интерфейс, могут использоваться объекты любого из этих двух классов, поскольку интерфейс для всех этих объектов остается одинаковым. Благодаря поддержке интерфейсов в C# может быть в полной мере реализован главный принцип полиморфизма: один интерфейс — множество методов.

Интерфейсы объявляются с помощью ключевого слова interface. Ниже приведена упрощенная форма объявления интерфейса:

```
interface имя {
    возвращаемый_тип имя_метода_1 (список_параметров);
    возвращаемый_тип имя_метода_2 (список_параметров);
    // ...
    возвращаемый_тип имя_метода_N (список_параметров);
}
```

где имя — это конкретное имя интерфейса.

В объявлении методов интерфейса используются только их возвращаемый\_тип и сигнатура. Они, по существу, являются абстрактными методами. Как пояснялось выше, в интерфейсе не может быть никакой реализации. Поэтому все методы интерфейса должны быть реализованы в каждом классе, включающем в себя этот интерфейс. В самом же интерфейсе методы неявно считаются открытыми, поэтому доступ к ним не нужно указывать явно.

Помимо методов, в интерфейсах можно также указывать свойства, индексаторы и события.

Интерфейсы не могут содержать члены данных. В них нельзя также определить конструкторы, деструкторы или операторные методы. Кроме того, ни один из членов интерфейса не может быть объявлен как `static`.

Как только интерфейс будет определен, он может быть реализован в одном или нескольких классах. Для реализации интерфейса достаточно указать его имя после имени класса, аналогично базовому классу. Ниже приведена общая форма реализации интерфейса в классе:

```
class имя_класса : имя_интерфейса {  
    // тело класса  
}
```

где `имя_интерфейса` — это конкретное имя реализуемого интерфейса. Если уж интерфейс реализуется в классе, то это должно быть сделано полностью. В частности, реализовать интерфейс выборочно и только по частям нельзя.

В классе допускается реализовывать несколько интерфейсов. В этом случае все реализуемые в классе интерфейсы указываются списком через запятую. В классе можно наследовать базовый класс и в тоже время реализовать один или более интерфейсов. В таком случае имя базового класса должно быть указано перед списком интерфейсов, разделяемых запятой.

Методы, реализующие интерфейс, должны быть объявлены как `public`. Дело в том, что в самом интерфейсе эти методы неявно подразумеваются как открытые, поэтому их реализация также должна быть открытой. Кроме того, возвращаемый тип и сигнатура реализуемого метода должны точно соответствовать возвращаемому типу и сигнатуре, указанным в определении интерфейса.

Пример программы.

#### 1. Листинг программы:

```
using System;  
  
using System.Collections.Generic;  
  
using System.Linq;  
  
using System.Text;  
  
namespace laba4  
{  
    public class Termin: IEquatable<Termin>  
    {  
        public string slovo { get; set; }  
        public int chastota { get; set; }  
  
        public bool Equals(Termin other)  
        {  
            if (Object.ReferenceEquals(other, null)) return false;
```

```

if (Object.ReferenceEquals(this, other)) return true;

return chastota.Equals(other.chastota) && slovo.Equals(other.slovo);

    }

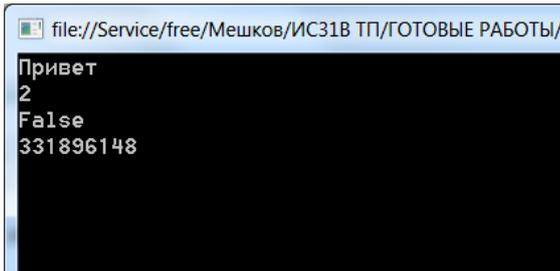
    public override int GetHashCode()
    {
        int hashTerminslovo = slovo == null ? 0 : slovo.GetHashCode();
        int hashTerminchastota = chastota.GetHashCode();
        return hashTerminslovo ^ hashTerminchastota;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Termin myClass = new Termin();
        myClass.slovo = "Привет";
        myClass.chastota = 2;
        //myClass.Equals(myClass.slovo, "Привет");
        Console.WriteLine(myClass.slovo);
        Console.WriteLine(myClass.chastota);
        Console.WriteLine(myClass.Equals("Привет", "Привет"));
        Console.WriteLine(myClass.GetHashCode());

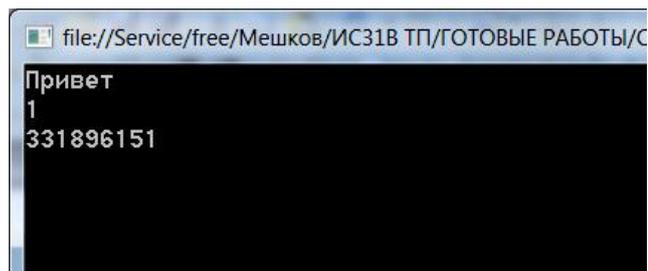
        Console.ReadLine();
    }
}
}

```

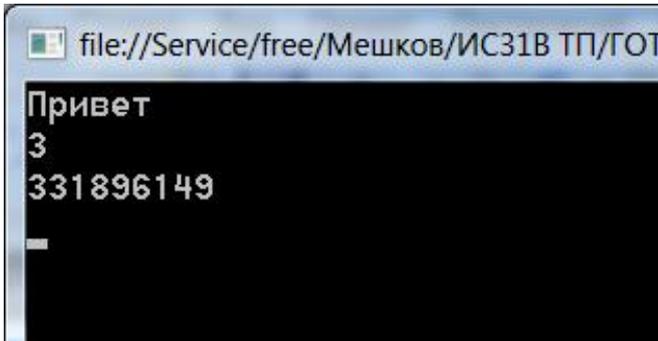
2. Пример окна рабочей программы:



```
file:///Service/free/Мешков/ИС31В ТП/ГОТОВЫЕ РАБОТЫ/  
Привет  
2  
False  
331896148
```



```
file:///Service/free/Мешков/ИС31В ТП/ГОТОВЫЕ РАБОТЫ/  
Привет  
1  
331896151
```



```
file:///Service/free/Мешков/ИС31В ТП/ГОТ  
Привет  
3  
331896149  
_
```

## Практическая работа №7. Групповые интерфейсы.

Цель работы: Получить навыки в программировании групповых интерфейсов для создания сценариев вызова методов.

Задание: Написать программу на языке C# для реализации групповых интерфейсов `IEquatable` для создания сценариев вызова методов. Методы и объекты создавать для прикладной области, указанной индивидуально преподавателем.

Теоретический материал.

Как это ни покажется странным, но в C# допускается объявлять переменные ссылочного интерфейсного типа, т.е. переменные ссылки на интерфейс. Такая переменная может ссылаться на любой объект, реализующий ее интерфейс. При вызове метода для объекта посредством интерфейсной ссылки выполняется его вариант, реализованный в классе данного объекта. Этот процесс аналогичен применению ссылки на базовый класс для доступа к объекту производного класса.

Переменной ссылки на интерфейс доступны только методы, объявленные в ее интерфейсе. Поэтому интерфейсную ссылку нельзя использовать для доступа к любым другим переменным и методам, которые не поддерживаются объектом класса, реализующего данный интерфейс.

Аналогично методам, свойства указываются в интерфейсе вообще без тела. Ниже приведена общая форма объявления интерфейсного свойства:

```
// Интерфейсное свойство
тип имя{
    get;
    set;
}
```

Очевидно, что в определении интерфейсных свойств, доступных только для чтения или только для записи, должен присутствовать единственный аксессор: `get` или `set` соответственно.

Несмотря на то что объявление свойства в интерфейсе очень похоже на объявление [автоматически реализуемого свойства](#) в классе, между ними все же имеется отличие. При объявлении в интерфейсе свойство не становится автоматически реализуемым. В этом случае указывается только имя и тип свойства, а его реализация предоставляется каждому реализующему классу. Кроме того, при объявлении свойства в интерфейсе не разрешается указывать модификаторы доступа для аксессоров. Например, аксессор `set` не может быть указан в интерфейсе как `private`.

Один интерфейс может наследовать другой. Синтаксис наследования интерфейсов такой же, как и у классов. Когда в классе реализуется один интерфейс, наследующий другой, в нем должны быть реализованы все члены, определенные в цепочке наследования интерфейсов.

Таким образом, интерфейсы могут быть организованы в иерархии. Как и в иерархии классов, в иерархии интерфейсов, когда какой-то интерфейс расширяет существующий, он наследует все абстрактные члены своего родителя (или родителей). Конечно, в отличие от классов, производные интерфейсы никогда не наследуют саму реализацию. Вместо этого они просто расширяют собственное определение за счет добавления дополнительных абстрактных членов.

Пример программы.

Модификатор `override` (ПЕРЕОПРЕДЕЛЕНИЕ) требуется для расширения или изменения абстрактной или виртуальной реализации унаследованного метода, свойства, индекса или события.

Метод **`override`** предоставляет новую реализацию члена, унаследованного от базового класса. Метод, переопределенный объявлением **`override`**, называется переопределенным базовым методом. Переопределенный базовый метод должен иметь ту же сигнатуру, что и метод **`override`**.

## 1. ЛИСТИНГ:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Worker : IEquatable<Worker>
    {
        public string fio;
        public string stat;

        public bool Equals(Worker other)
        {
            if (Object.ReferenceEquals(other, null)) return false;
            if (Object.ReferenceEquals(this, other))
                return true;
            return fio.Equals(other.fio) && stat.Equals(other.stat);
        }

        public override bool Equals(Object obj)
        {
            return Equals(obj as Worker);
        }

        public override int GetHashCode()
        {
            return stat.GetHashCode() ^ fio.GetHashCode();
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Worker ich = new Worker();
            ich.fio= "ИВАНОВ";
            ich.stat= "СТУДЕНТ";

            Console.WriteLine(ich.fio);
            Console.WriteLine(ich.stat);
            Console.WriteLine(ich.GetHashCode());
            Console.ReadLine();
        }
    }
}
```

2.

file://Service/free/Мешков/  
Иванов  
студент  
-1556237025

file://Service/free/Мешк  
Петров  
студент  
1232616648  
-

## Практическая работа №8. Списки.

**Цель работы:** Освоить создание списка и элементов управления им в среде MVS, используя стандартный класс списка.

**Задание:** Составить небольшую программу в MVS, которая будет представлять список, и основные элементы управления – добавление, удаление и поиск элементов.

Пример программы.

1. Листинг

```
using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        List<string> list = new List<string>();

        string a;

        string b;

        string c;

        public int d = new int();

        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
```

```

try
{
    if (textBox1.Text == "")
    {
        MessageBox.Show("Введите любые данные в поле ввода.\n", "Ошибка!", MessageBoxButtons.OK,
        MessageBoxIcon.Error);

        return;
    }
    else
    {
        a = Convert.ToString(textBox1.Text);
        list.Add(a);
        textBox1.Clear();

        //

        dataGridView1.Rows.Clear();
        foreach (string val in list)
        {
            dataGridView1.Rows.Add(val);
        }
    }
}
catch (FormatException)
{
    MessageBox.Show("Введите любые данные в поле ввода.\n", "Ошибка!", MessageBoxButtons.OK,
    MessageBoxIcon.Error);

    return;
}
}

private void button2_Click(object sender, EventArgs e)
{

```

```

try
{
    if (radioButton1.Checked == false && radioButton2.Checked == false)
    {
        MessageBox.Show("Укажите, по какому критерию необходимо удалить элемент.\n", "Вопрос.",
        MessageBoxButtons.OK, MessageBoxIcon.Warning);

        return;
    }
    else
    {
        if (radioButton1.Checked == true)
        {
            d = Convert.ToInt32(textBox4.Text);
            list.RemoveAt(d - 1);
            textBox4.Clear();

            dataGridView1.Rows.Clear();

            foreach (string val in list)
            {
                dataGridView1.Rows.Add(val);
            }
            dataGridView1.Rows.Clear();
            foreach (string val in list)
            {
                dataGridView1.Rows.Add(val);
            }
        }
        else
        {
            if (textBox2.Text == "")
            {
                MessageBox.Show("Введите элемент для удаления.\n", "Вопрос.", MessageBoxButtons.OK,
                MessageBoxIcon.Warning);
            }
        }
    }
}

```

```

        return;
    }

    else
    {
        b = Convert.ToString(textBox2.Text);

        list.Remove(b);

        textBox2.Clear();

                                dataGridView1.Rows.Clear();

        foreach (string val in list)
        {
                                dataGridView1.Rows.Add(val);
        }

    }
}
}
}
}

catch (FormatException)
{
    MessageBox.Show("Введите любое целое число в поле ввода номера элемента или значение в поле
ввода элемента .\n", "Ошибка!", MessageBoxButtons.OK, MessageBoxIcon.Error);

    return;
}

catch (ArgumentOutOfRangeException)
{
    MessageBox.Show("В списке отсутствует элемент под указанным номером.\n", "Ошибка!",
MessageBoxButtons.OK, MessageBoxIcon.Error);

    return;
}
}

private void button3_Click(object sender, EventArgs e)
{

```

```

try
{
    c = Convert.ToString(textBox3.Text);

    if (list.Contains(c))
    {
        MessageBox.Show("Элемент найден.\n", "Результат.", MessageBoxButtons.OK,
        MessageBoxIcon.Warning);

        return;
    }

    else
    {
        MessageBox.Show("Элемент не найден.\n", "Ошибка!", MessageBoxButtons.OK,
        MessageBoxIcon.Error);

        return;
    }
}

catch (FormatException)
{
    MessageBox.Show("Введите любое целое число в поле ввода элемента.\n", "Ошибка!",
    MessageBoxButtons.OK, MessageBoxIcon.Error);

    return;
}

catch (ArgumentOutOfRangeException)
{
    MessageBox.Show("В списке отсутствует элемент под указанным номером.\n", "Ошибка!",
    MessageBoxButtons.OK, MessageBoxIcon.Error);

    return;
}
}

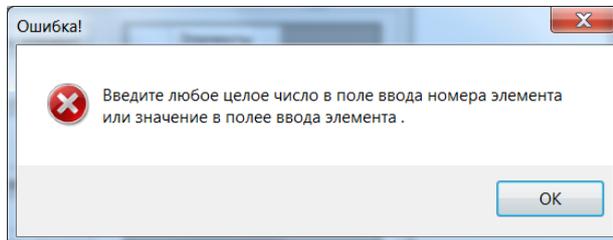
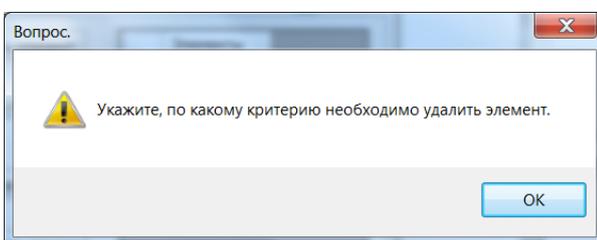
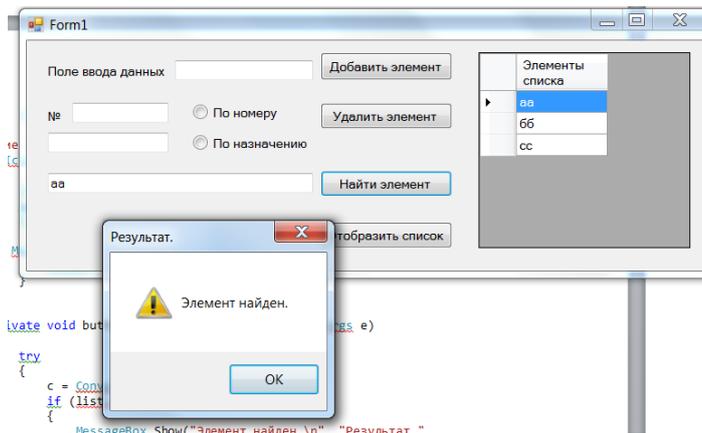
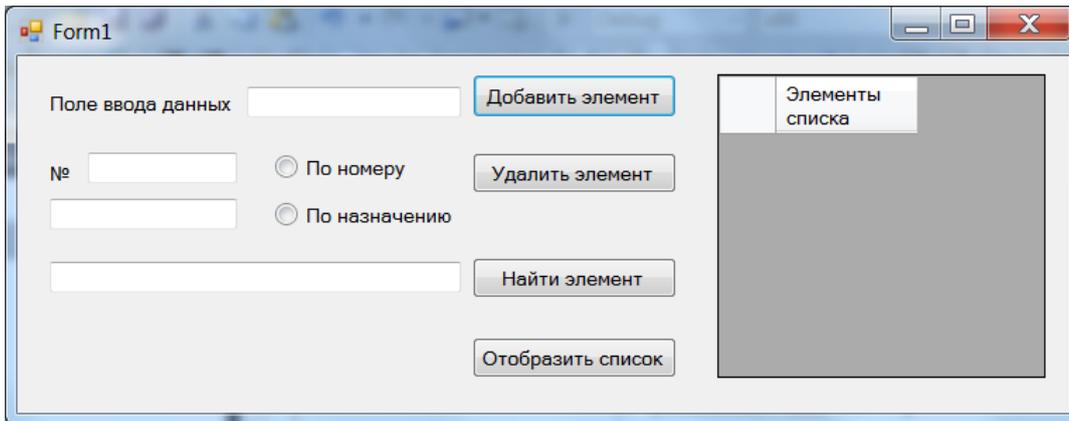
private void button4_Click(object sender, EventArgs e)
{
    dataGridView1.Rows.Clear();

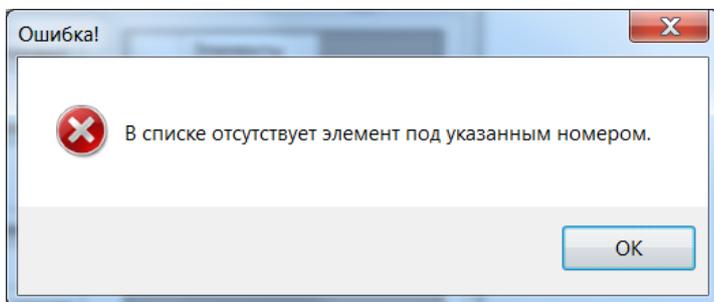
    foreach (string val in list)

```

```
{
    dataGridView1.Rows.Add(val);
}
}
}
}
```

## 2. Примеры окна программы:





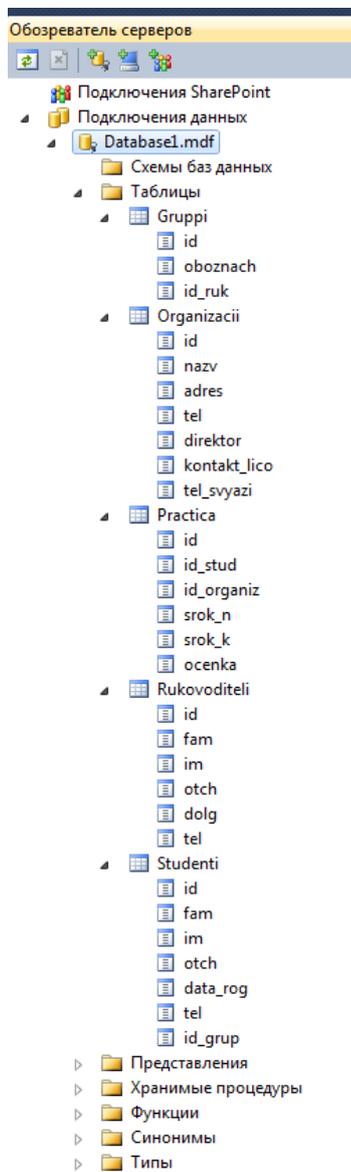
## Практическая работа №9. Работа с базами данных SQL.

**Цель работы:** Освоить работу с SQL базами.

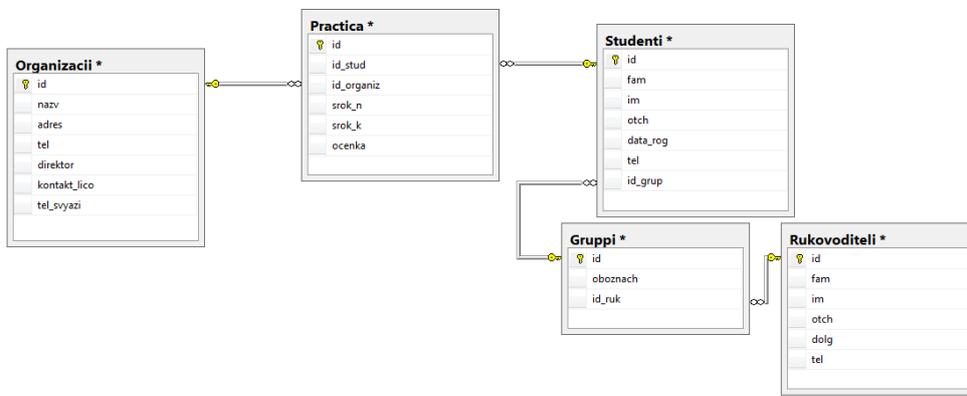
**Задание:** Написать программу работающую с базами данных SQL. Объекты прикладной области для базы данных задаются преподавателем индивидуально.

**Ход работы:**

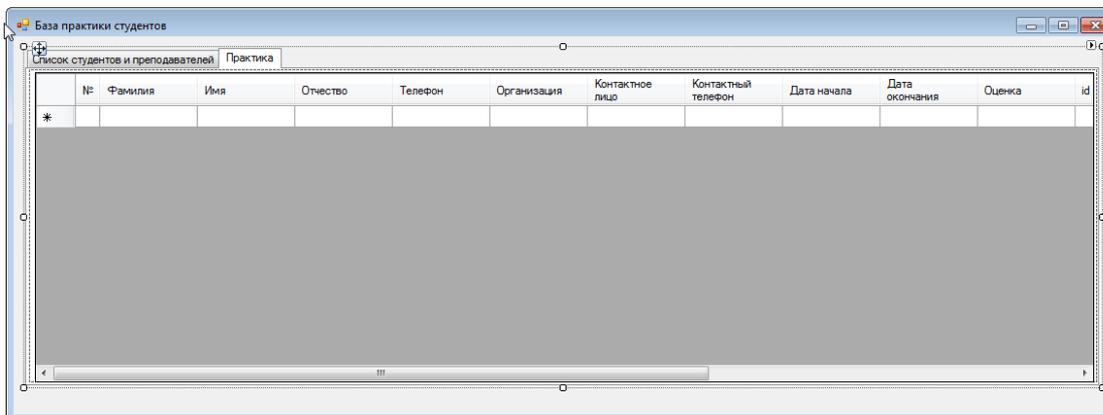
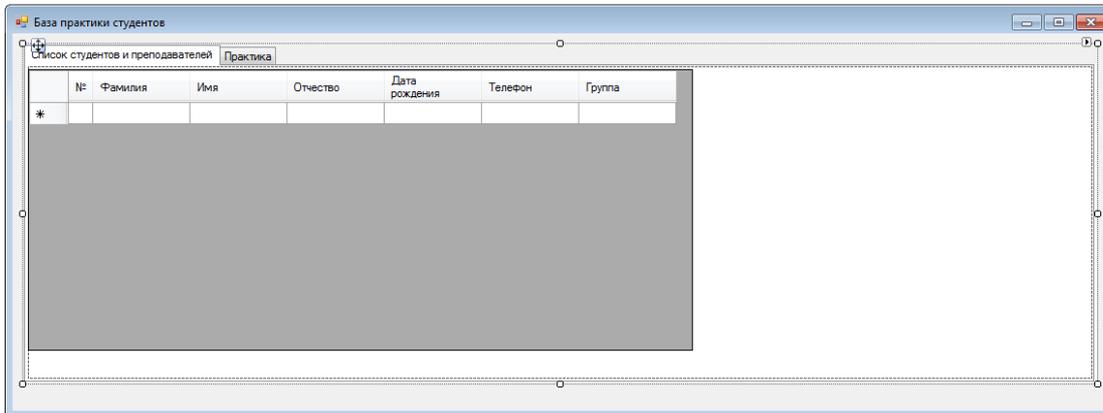
1. Создаем базу данных следующей структуры:



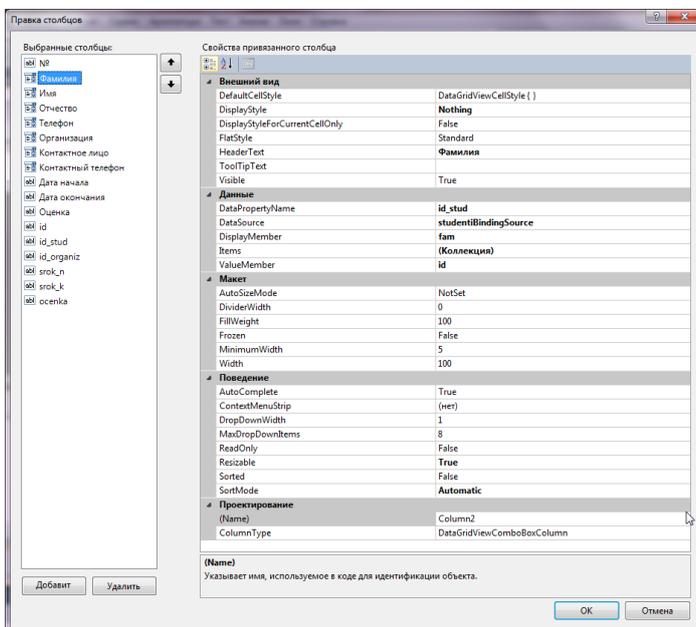
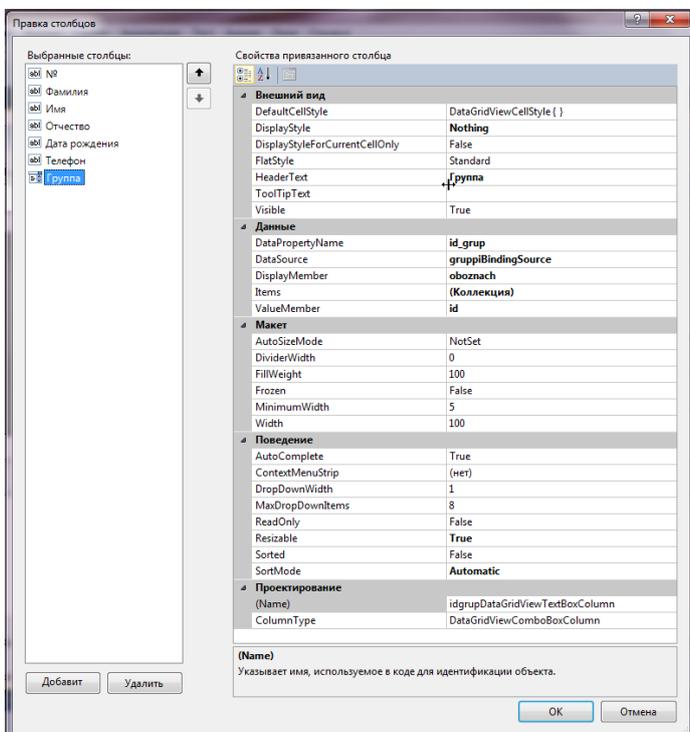
2. Определяем связи между таблицами:



### 3. Создаем интерфейс:



### 4. Настраиваем подключение базы данных к DataGrid'у и в нем настраиваем столбцы:



## 5. Листинг программы:

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.OleDb;

namespace laba1
{

```

```

public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        // TODO: данная строка кода позволяет загрузить данные в таблицу "database1DataSet5.Organizacii".
        При необходимости она может быть перемещена или удалена.
        this.organizaciiTableAdapter.Fill(this.database1DataSet5.Organizacii);
        // TODO: данная строка кода позволяет загрузить данные в таблицу "database1DataSet4.Practica". При
        необходимости она может быть перемещена или удалена.
        this.practicaTableAdapter.Fill(this.database1DataSet4.Practica);
        // TODO: данная строка кода позволяет загрузить данные в таблицу "database1DataSet3.Gruppi". При
        необходимости она может быть перемещена или удалена.
        this.gruppiTableAdapter.Fill(this.database1DataSet3.Gruppi);
        // TODO: данная строка кода позволяет загрузить данные в таблицу "database1DataSet2.Studenti". При
        необходимости она может быть перемещена или удалена.
        this.studentiTableAdapter.Fill(this.database1DataSet2.Studenti);
    }
}

```

## 6. Готовое окно программы:

База практики студентов

Список студентов и преподавателей | Практика

№	Фамилия	Имя	Отчество	Дата рождения	Телефон	Группа
1	Пиманский	Владимир	Васильевич	07.11.1996		ИС31В
2	Махнев	Виталий	Владимирович	28.09.1996		ИС31В
3	Симонов	Александр	Сергеевич	07.12.1995		ИС31В
4	Скобелева	Руслана	Андреевна	18.12.1994	9034325490	ИС31В
5	Слуцкий	Андрей	Валерьевич	07.10.1996		ИС31В
6	Тушков	Максим	Витальевич	02.06.1996		ИС31В
7	Худин	Алексей	Александрович	03.07.1994		ИС31В
8	Якимова	Ирина	Сергеевна	19.05.1996		ИС31В
*						

База практики студентов

Список студентов и преподавателей | Практика

№	Фамилия	Имя	Отчество	Телефон	Организация	Контактное лицо	Контактный телефон	Дата начала	Дата окончания	Оценка
1	Пиманский	Владимир	Васильевич		ОАО Атоммаш...	Олегова В.Я.	23-56-89	12.12.2016	25.01.2016	хорошо
2	Махнев	Виталий	Владимирович		ОАО Атоммаш...	Олегова В.Я.	23-56-89	12.12.2016	25.01.2016	отлично
3	Симонов	Александр	Сергеевич		Волгодонский х...	Иванченко Т.Д.	98-54-21	12.12.2016	25.01.2016	хорошо
4	Скобелева	Руслана	Андреевна	9034325490	Волгодонский х...	Иванченко Т.Д.	98-54-21	12.12.2016	25.01.2016	отлично
5	Слуцкий	Андрей	Валерьевич		ООО Компьюте...	Иванова Я.К.	11-22-33	12.12.2016	25.01.2016	отлично
6	Тушков	Максим	Витальевич		ООО Компьюте...	Иванова Я.К.	11-22-33	12.12.2016	25.01.2016	хорошо
7	Худин	Алексей	Александрович		ОАО Атоммаш...	Олегова В.Я.	23-56-89	12.12.2016	25.01.2016	хорошо
8	Якимова	Ирина	Сергеевна		Волгодонский х...	Иванченко Т.Д.	98-54-21	12.12.2016	25.01.2016	удовлетворител...
*										

## Практическая работа №10. Регулярные выражения.

**Цель:** Научиться выделять структурированную информацию из произвольного текста с использованием регулярных выражений.

**Задание:** Составить небольшую программу в MVS, которая будет искать строки, соответствующие заданному регулярному выражению. Выделяемая структура задается преподавателем индивидуально.

### Теоретический материал.

Регулярные выражения — это часть небольшой технологической области, невероятно широко используемой в огромном диапазоне программ. Регулярные выражения можно представить себе как мини-язык программирования, имеющий одно специфическое назначение: находить подстроки в больших строковых выражениях.

Регулярные выражения поддерживаются множеством классов .NET из пространства имен System.Text.RegularExpressions.

Язык регулярных выражений предназначен специально для обработки строк. Он включает два средства:

Набор управляющих кодов для идентификации специфических типов символов

Система для группирования частей подстрок и промежуточных результатов таких действий

С помощью регулярных выражений можно выполнять достаточно сложные и высокоуровневые действия над строками:

Идентифицировать (и возможно, пометить к удалению) все повторяющиеся слова в строке

Сделать заглавными первые буквы всех слов

Преобразовать первые буквы всех слов длиннее трех символов в заглавные

Обеспечить правильную капитализацию предложений

Выделить различные элементы в URI (например, имея `http://www.professorweb.ru`, выделить протокол, имя компьютера, имя файла и т.д.)

Главным преимуществом регулярных выражений является использование метасимволов — специальные символы, задающие команды, а также управляющие последовательности, которые работают подобно управляющим последовательностям C#. Это символы, предваренные знаком обратного слеша (\) и имеющие специальное назначение.

В следующей таблице специальные метасимволы регулярных выражений C# сгруппированы по смыслу:

Метасимволы, используемые в регулярных выражениях C#

Символ	Значение	Пример	Соответствует
Классы символов			

[...]	Любой из символов, указанных в скобках	[a-z]	В исходной строке может быть любой символ английского алфавита в нижнем регистре
[^...]	Любой из символов, не указанных в скобках	[^0-9]	В исходной строке может быть любой символ кроме цифр
.	Любой символ, кроме перевода строки или другого разделителя Unicode-строки		
\w	Любой текстовый символ, не являющийся пробелом, символом табуляции и т.п.		
\W	Любой символ, не являющийся текстовым символом		
\s	Любой пробельный символ из набора Unicode		
\S	Любой непробельный символ из набора Unicode. Обратите внимание, что символы \w и \S - это не одно и то же		
\d	Любые ASCII-цифры. Эквивалентно [0-9]		
\D	Любой символ, отличный от ASCII-цифр. Эквивалентно [^0-9]		

##### 5. @"b(http://w+.\w+)"

```

file:///E:/Скобелева Р.А/Проекты/laba6 (regulars)/ConsoleApplication1/ConsoleApplication1/bin/D...
Регистрозависимый поиск:
В исходной строке: "searchlink": "http://vis2.ru/index.php?option=com_search&view=search&tmpl=component" есть совпадения!
В исходной строке: "uribase": "http://vis2.ru/" есть совпадения!

Регистронезависимый поиск:
В исходной строке: "searchlink": "http://vis2.ru/index.php?option=com_search&view=search&tmpl=component" есть совпадения!
В исходной строке: "adusearchlink": "HTTP://vis2.ru/index.php?option=com_search&view=search" есть совпадения!
В исходной строке: "uribase": "http://vis2.ru/" есть совпадения!

http://vis2.ru

```

Ход работы.

1. Найдем все гиперссылки в данном наборе строк:

```
"page",
"Страница",
"page_of: ' of", "searchlink':
"http://vis2.ru/index.php?option=com_search&view=search&tmpl=component",
"advsearchlink': 'HTTP://vis2.ru/index.php?option=com_search&view=search",
"uribase': 'http://vis2.ru/",
"limit': '50",
"perpage': '5",
"rdering': 'popular",
"phrase': 'any",
"hidedivs': "",
"includelink': 0",
"viewall': ' Просмотреть все результаты"
```

2. ЛИСТИНГ

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            // Массив тестируемых строк
            string[] test = {
```

```

    "page",
        "Страница",
        "page_of": " " of", "searchlink":
"http://vis2.ru/index.php?option=com_search&view=search&tmpl=component",
        "advsearchlink": 'HTTP://vis2.ru/index.php?option=com_search&view=search',
        "uribase": 'http://vis2.ru/',
        "limit": '50',
        "perpage": '5',
        "rdering": 'popular',
        "phrase": 'any',
        "hidedivs": "",
        "includelink": '0',
        "viewall": 'Просмотреть все результаты'
};

```

```

// Проверим, содержится ли в исходных строках слово World
// при этом мы не укажем опции RegexOptions
Regex regex = new Regex("http://");

Console.WriteLine("Регистрозависимый поиск: ");
foreach (string str in test)
{
    if (regex.IsMatch(str))
        Console.WriteLine("В исходной строке: \"{0}\" есть совпадения!", str);
}
Console.WriteLine();

// Теперь укажем поиск, не зависящий от регистра
regex = new Regex("http://", RegexOptions.IgnoreCase);

Console.WriteLine("РегистроНЕзависимый поиск: ");

```



```

Regex Red = new Regex(@"\w+|\@|\w+\.|\w+");
Match cool = Red.Match(text); //коллекция совпадения
while (cool.Success)
{
    Console.WriteLine("Почта");
    Console.WriteLine(cool.Value);
    cool = cool.NextMatch();
}

Console.WriteLine("");

Regex Red2 = new Regex(@"www[^\s;""']+|http[^\s;""']+");
Match cool2 = Red2.Match(text);
while (cool2.Success)
{
    Console.WriteLine("Ссылка");
    Console.WriteLine(cool2.Value);
    cool2 = cool2.NextMatch();
}

Console.WriteLine("");

Regex Red3 = new Regex(@"\+7.+|(\d.+");
Match cool3 = Red3.Match(text);
while (cool3.Success)
{
    Console.WriteLine("Телефон");
    Console.WriteLine(cool3.Value);
    cool3 = cool3.NextMatch();
}

```



## Практическая работа №11. Поиск совпадений в объектах методами IComparable

### Теоретический материал.

Если требуется отсортировать коллекцию, состоящую из объектов определяемого пользователем класса, при условии, что они не сохраняются в коллекции класса `SortedList`, где элементы располагаются в отсортированном порядке, то в такой коллекции должен быть известен способ сортировки содержащихся в ней объектов. С этой целью можно, в частности, реализовать интерфейс `IComparable` для объектов сохраняемого типа. Интерфейс `IComparable` доступен в двух формах: обобщенной и необобщенной. Несмотря на сходство применения обеих форм данного интерфейса, между ними имеются некоторые, хотя и небольшие, отличия.

Если требуется отсортировать объекты, хранящиеся в необобщенной коллекции, то для этой цели придется реализовать необобщенный вариант интерфейса `IComparable`. В этом варианте данного интерфейса определяется только один метод, `CompareTo()`, который определяет порядок выполнения самого сравнения.

Ниже приведена общая форма объявления метода `CompareTo()`:

```
int CompareTo(object obj)
```

В методе `CompareTo()` вызывающий объект сравнивается с объектом `obj`. Для сортировки объектов по нарастающей конкретная реализация данного метода должна возвращать нулевое значение, если значения сравниваемых объектов равны; положительное — если значение вызывающего объекта больше, чем у объекта `obj`; и отрицательное — если значение вызывающего объекта меньше, чем у объекта `obj`. А для сортировки по убывающей можно обратить результат сравнения объектов. Если же тип объекта `obj` не подходит для сравнения с вызывающим объектом, то в методе `CompareTo()` может быть сгенерировано исключение `ArgumentException`.

Пример. Для удобства создан пользовательский класс *Termin* с полями: слово, частота. Пользовательский класс прописан с обобщенным методом сравнения *IComparable*, для возможности сравнивать объекты пользовательских классов.

```
class Termin : IComparable
{
    public string slovo;
    public int chastota;
    public int TF1;// относительная доля слова
        public int TF2;

    int IComparable.CompareTo(object obj)
    {
```

```

Termin Compare Obj = obj as Termin;

if (Compare Obj == null) throw new ArgumentException("Можно сравнивать только элементы одного
типа.");

return slovo.Compare To(Compare Obj.slovo);
}

```

Это необходимо в случае использования в качестве ключей своих собственных классов, которые не поддерживают ни интерфейса *IComparable*, ни интерфейса *IComparable<K>*, или при желании сделать так, чтобы объекты сравнивались посредством какого-то нестандартного процесса [17, 21].

В данном случае нужно сравнение объекта класса по полю *slovo*, что и сделано:

```
return slovo.Compare To(Compare Obj.slovo);
```

Иногда, для сравнения и установления идентичности объектов, нужен хеш-код, тогда нужен и интерфейс *IHashCodeProvider*, который будет подробнее описан ниже.

Пример программы.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Woker : IComparable//интерфейс для сортировки в пользовательском классе
    {
        public string Surname;
        public string Name;
        public string SecName;
        public string job;
        public int payment;

        int IComparable.CompareTo(object obj)//прописываем метод CompareTo(сравнение) (возвращает int - если
0 то объекты равны, если 1 то больше, если -1 то меньше)
        {
            Woker CompareObj = obj as Woker;
            if (CompareObj == null) throw new ArithmeticException("pz");
            //return Surname.CompareTo(CompareObj.Surname);//по какому полю мы сравниваем
            return payment.CompareTo(CompareObj.payment);
            //return job.CompareTo(CompareObj.job);
        }
    }
}

```

```

static void Main(string[] args)
{
    List<Woker> Personal = new List<Woker> { };//создаем список с элементами класса Woker
    Woker w1 = new Woker();

    w1.Surname = "DEXP M182";
    w1.Name = "Диагональ экрана 18.5";
    w1.SecName = "Максимальное разрешение 1366x768";
    w1.payment = 4999 ;
    Personal.Add(w1);

    Woker w2 = new Woker();

    w2.Surname = "Диагональ экрана 20";
    w2.Name = "Максимальное разрешение 1600x900";
    w2.SecName = "Максимальное разрешение 1366x768";
    w2.payment = 4800 ;
    Personal.Add(w2);

    Woker w3 = new Woker();

    w3.Surname = "Lenovo LI2215sD";
    w3.Name = "Диагональ экрана 21.5";
    w3.SecName = "Максимальное разрешение 1920x1080";
    w3.payment = 5500 ;
    Personal.Add(w3);

    foreach (Woker p in Personal)//вывели лист до сортировки
    { Console.WriteLine(p.Surname + " " + p.Name + " " + p.SecName + " " + p.job + " " + p.payment); }
    Personal.Sort();//сортировка с исп. компаратора который мы прописали вначале
    Console.WriteLine("_____");
    foreach (Woker p in Personal)//вывели лист после сортировки
    { Console.WriteLine(p.Surname + " " + p.Name + " " + p.SecName + " " + p.job + " " + p.payment); }

    Console.ReadLine();
}
}
}

```

```
file://Service/free/Мешков/ИСЗ1В ИСИС/Лиманский/lab4/ConsoleApplication1/ConsoleApplication1/bi...
DEXP M182 Диагональ экрана      18.5 Максимальное разрешение   1366x768 4999
Диагональ экрана      20 Максимальное разрешение   1600x900 Максимальное ра
зрешение      1366x768 4800
Lenovo L12215sD Диагональ экрана      21.5 Максимальное разрешение   1920x108
0 5500

Диагональ экрана      20 Максимальное разрешение   1600x900 Максимальное ра
зрешение      1366x768 4800
DEXP M182 Диагональ экрана      18.5 Максимальное разрешение   1366x768 4999
Lenovo L12215sD Диагональ экрана      21.5 Максимальное разрешение   1920x108
0 5500
```