Лабораторная работа № 2

Среда программирования Microsoft Visual C# 2010 Express

Цель работы: Изучить возможности среды программирования Visual C#. Приобрести навыки отладки линейных программ.

Отчет по работе должен содержать:

Постановку задачи, программный код (скопировать из среды как текст с форматированием), и протокол работы по каждому из двух заданий:

1) **Индивидуальное задание 1 (с.10)**
2) **Индивидуальное задание 2 (с.23)**

Предварительно прочитать и выполнить описанные (разобранные) примеры!!!!!!

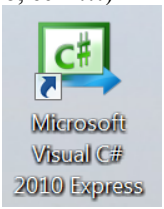
Задание 1

Назначение среды программирования.

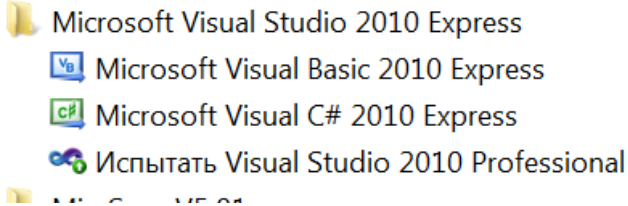
Среда программирования Visual Studio предназначена для разработки программ на различных языках высокого уровня. Используя эту среду, можно разрабатывать программы на таких языках, как Basic, C#, C++, J#. В данной лабораторной работе рассматривается среда языка C#. Вместо термина «программа» принято использовать термин «приложение» или «проект». Этот термин более точно отражает суть того, что создает программист. Результат его работы – это программный файл, использующий возможности встроенных библиотек операционной системы Windows и библиотек самой среды программирования. Создаваемое в результате разработки программное средство может работать только при наличии библиотек среды и не является самостоятельным.

Запуск среды.

Работу в среде можно начать одним из двух способов:

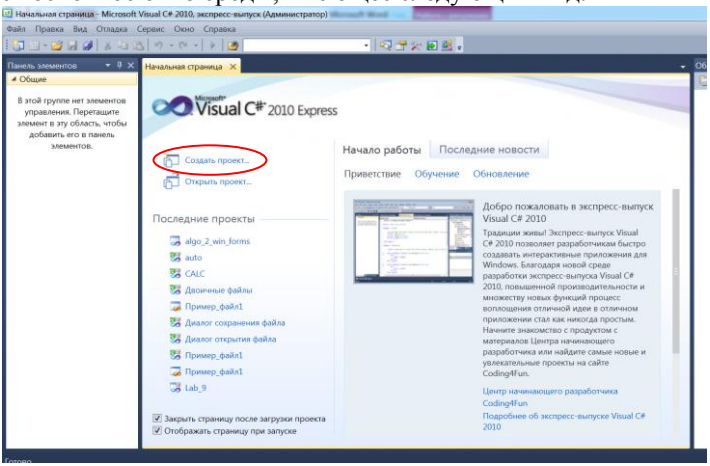
- Запустить среду с помощью значка на рабочем столе Windows (если он там, конечно, есть…)
- 

- Запустить среду через кнопку ***«ПУСК»*** в меню ***«Программы:***



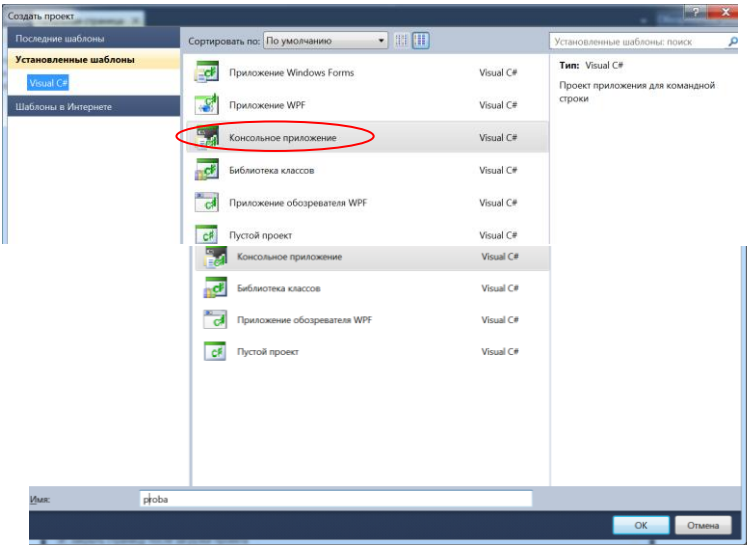
Начало работы.

Воспользуемся для запуска среды любым способом. На экране появится основное окно среды, имеющее следующий вид:



Выберем пункт «Создать проект», появится окно выбора вида проекта. Мы будем разрабатывать учебные программы, работающие в среде Windows, но как **консольные приложения** (имитация среды DOS). Выбор такого варианта обучения обусловлен тем, чтобы изучению основных приемов алгоритмизации не мешали особенности организации интерфейса при программирования под Windows – эти особенности вы будете изучать позднее, когда с вопросами разработки алгоритмов у вас не будет проблем. А пока наберитесь терпения – все проекты мы будем строить в виде консольных приложений.

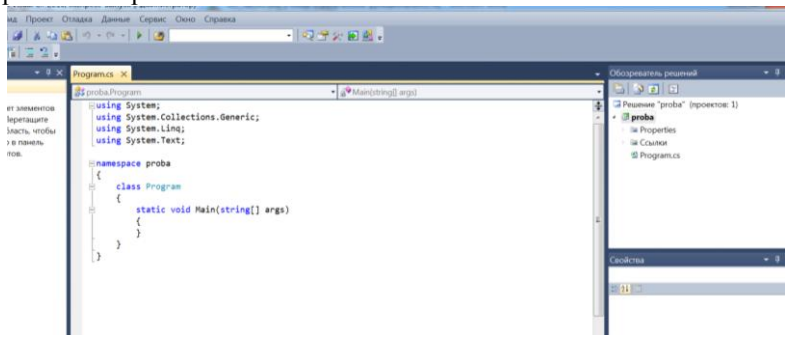
Выберем «Консольное приложение»:



В поле Имя (внизу) наберем имя проекта, например ***Proba***:

Желательно использовать в качестве имен проекта и имен программных модулей идентификаторы, составленные из латинских букв (не использовать русские) – так вы избежите некоторых проблем при запуске и исполнении программы.

Нажимаем кнопку Ok. Начинает работать мастер создания проекта. Его работа завершается появлением окна:

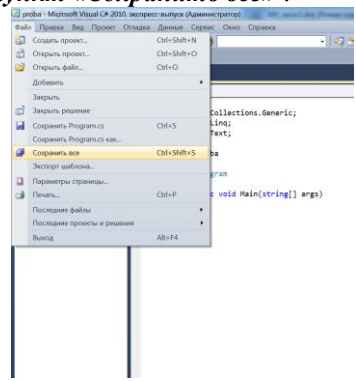


В окне **Обозреватель решений** виден состав нашего проекта. Как видим, он не пуст – в папках уже кое-что имеется. Мастер проекта в среде C# уже поместил в них необходимые компоненты. Именно те, которые достаточны для организации консольного приложения. Сущность компо-

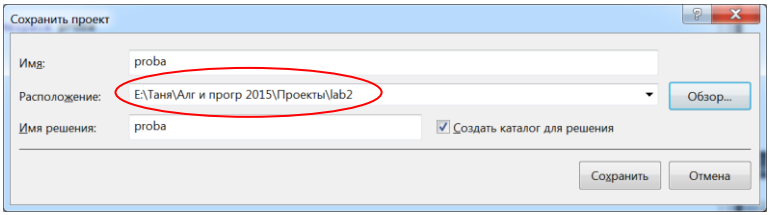
нент вы постепенно изучите и поймете. Сейчас рассмотрим только то, что необходимо понимать на начальной стадии обучения.

Прежде всего, отметим, что вся наша разработка – это **решение.** Именно такой термин принят при разработке программных систем в этой среде. Каждое решение в C# может содержать несколько **проектов**. В нашем случае – это один проект под названием ***proba.*** В составе проекта уже имеется один программный файл – ***Program.cs***. Тип файла (***cs***, сишарп) определяет его как файл, содержащий код на языке C#. Содержимое этого файла отображено в правом окне.

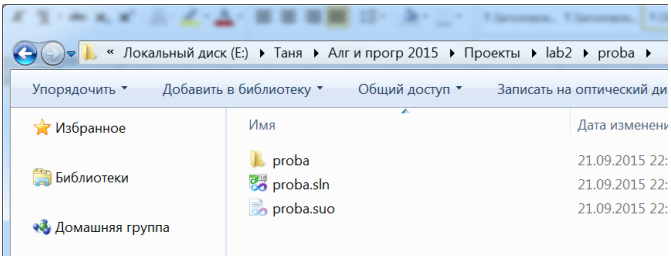
Пока ничего не набирая в тексте программного кода, сохраним проект – ***выберем в меню пункт «Сохранить все»*** :



Выберем папку с месторасположением проекта:

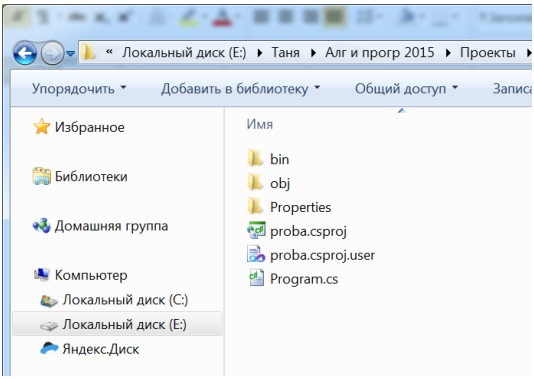


Сохраним, полностью закроем среду и попробуем отыскать, где именно на жестком диске разместилось наше решение. Через ***Мой компьютер*** откроем папку ***proba.*** Раскроем ее. Внутри увидим еще одну папку ***proba*** и пару файлов:



Один из файлов с расширением ***sln*** содержит информацию о нашем решении. Его можно использовать как запускающий файл. Дважды щелкнем левой кнопкой мышки – решение вновь раскроется.

Вновь закроем окно решения и продолжим рассмотрение папки ***proba***. Раскроем вложенную папку ***proba.*** Внутри увидим несколько папок и файлов:



Файл с расширением ***csproj*** – это запускающий файл проекта. Дважды щелкнем по нему и вновь откроется полное решение.

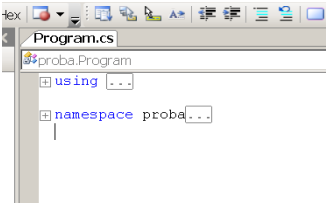
Вновь закроем окно решения. Обратим внимание на другой файл, у которого расширение ***cs***. Это тот самый файл, который содержит код на языке C#. Попробуем использовать его в качестве стартующего – дважды щелкнем его мышкой. Обратите внимание – открылся только сам файл, но не решение.

Закройте файл и вновь откройте решение. Перейдем к рассмотрению той заготовки, которую создал мастер проекта в файле ***Program.***

Содержание заготовки файла.

Прежде всего, обратите внимание на структуру информации в программном окне. Информация как бы разбита на несколько блоков. Каждый блок можно отобразить в развернутом (-) или сжатом (+) виде. Это

очень удобно, когда имеется большой программный текст. Часть можно свернуть, чтобы не мешал, а раскрыть только тот, который нужен. Например, если свернуть все блоки в нашем файле, то получится следующее:

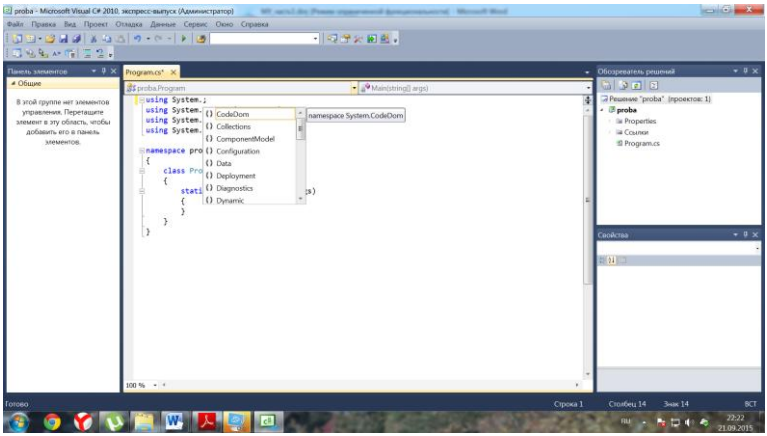


Развернем все блоки и рассмотрим каждый из них подробнее. В первом блоке описаны строки:

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

Ключевое слово ***using*** используется для описания системных возможностей, которые предоставлены в распоряжение программиста. Все языки программирования имеют системные возможности, и эти возможности оформлены в виде некоторых **системных библиотек**. В языке C# используется определенная терминология для описания вложенных возможностей. Не говорят о библиотеках возможностей – говорят о ***пространстве имен***. Каждая встроенная возможность имеет некоторое имя, а ряд близких по смыслу возможностей объединяется в группы. Получается, что ряд имен возможностей существует в некотором **пространстве** (***namespace***). Примером такого пространства является пространство ***System.*** Это основное пространство имен.

Ради любопытства поставьте в первой строке после слова System точку и посмотрите, что будет:



В выпавшем окне имеется еще множество уточнений, которые, в свою очередь, могут быть также другими пространствами имен или именами ***классов***, которые также объединяют в себе вложенные средства языка. Две другие строки этого блока используют подпространства имен пространства ***System.*** Описания пространств позволяют вызывать для исполнения встроенные возможности по их именам, потому что сами пространства уже описаны.

Перейдем к рассмотрению второго блока. Он называется так же, как и наш проект – ***proba.*** Предваряется словом ***namespace.*** Это значит, что все имена (идентификаторы), которые мы будем использовать в тексте, будут автоматически включены в пространство имен ***proba.*** Внутри пространства имен ***proba*** имеется описание класса с именем ***Program.*** В составе класса ***Program*** уже включен метод ***Main***. Это особый метод. Он всегда есть в любой программе, причем встречается он один-единственный раз. Это главный метод программы, что, собственно и означает слово «main». Кроме него в программе мы можем описать множество других методов, причем в самых разных классах.

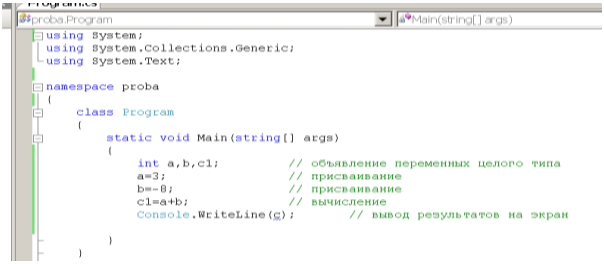
<u>Набор текста программы.</u>

Наберем в окне файла ***Program.cs*** текст программы. При наборе разные слова сами буду окрашиваться разным цветом. Итак, внутри метода ***Main*** набираем следующий текст:

```
    int a,b,c1;                    // объявление переменных
целого типа
    a=3;                           // присваивание
    b=-8;                          // присваивание
    c1=a+b;                        // вычисление
```

```
Console.WriteLine(c);                    // вывод результатов на экран
```

Получится:



Зеленый цвет используется средой для отображения ***комментариев***, синий – для ***служебных*** слов языка (их компилятор «понимает» только в том смысле, который в них заложен). Голубым цветом отображаются имена пространств и классов. Красный цвет (здесь его нет) – это цвет для отображения ***строк.*** Все остальное – на совести программиста.

Суть программы, которую вы набрали, достаточно проста. Объявляются три переменных целого типа. Две из них получают численные значения, а значение третьей вычисляется. Полученный результат выводится на экран, после чего программа завершает работу.

Подготовка программы к выполнению.

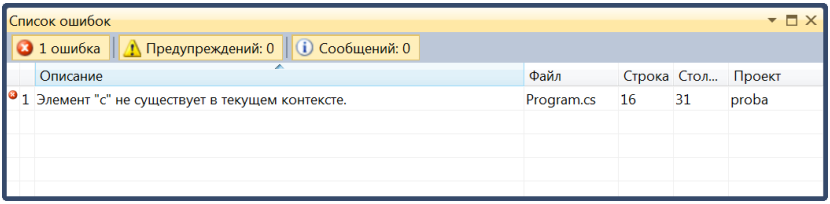
Среда предоставляет ряд возможностей, обеспечивающих как подготовку к запуску программы, так и немедленный запуск. Выполним предварительно небольшую настройку среды. Обратите внимание: сразу под меню размещается полоска панели инструментов – это стандартная панель. Чуть ниже размещена еще одна панель – это панель текстового редактора. Для работы очень удобно иметь еще одну панель инструментов – панель, которая называется ***Построение***. Для того, чтобы разместить ее на экране, проделаем следующее. Установив указатель мышки на стандартную панель, нажмем правую кнопку и выберем ***Построение*** и установим панель там, где вам покажется удобнее всего – например, рядом с панелью текстового редактора.

На панели всего три кнопки. Установим мышку на первую слева и прочитаем подсказку: ***Построить proba***. Кнопка позволяет запустить два процесса: сначала процесс **компиляции** программы, а затем процесс **создания** полного проекта. Компиляция – это перевод текста с языка C# в двоичный код. В процессе компиляции выполняется проверка правильности текста программы с точки зрения языка программирования – так называемый синтаксический контроль.**.** Если в процессе компиляции не обнаружится ошибок, то запускается второй процесс, который програм-

мисты называют «**линкование**». Смысл процесса - организация связей между отдельными компонентами проекта и динамическими библиотеками операционной системы. На этом этапе также могут быть обнаружены ошибки.

Теперь установим мышку на среднюю кнопку и прочитаем надпись: ***Построить решение.*** Кнопка запускает сначала процесс ***Построить proba,*** а затем выполняет ряд операций связанных с решением, т.е. здесь выполняется обработка всех проектов решения. Для нашего случая и для большинства консольных приложений действие этой кнопки совпадает с действием предыдущей кнопки.

Правая кнопка панели (***Отмена***) тоже может пригодиться. С помощью этой кнопки всегда можно прервать выполнение какого-то процесса или даже выполнение программы. Теперь проверим нашу программу. Сначала выполним компиляцию, нажав соответствующую кнопку. Получим:

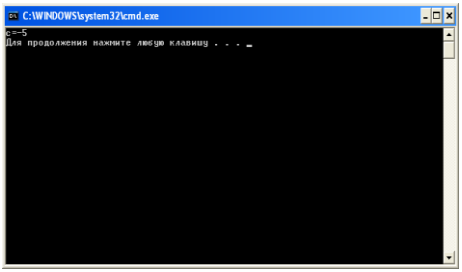


Это сообщение об ошибке в строке программы: в программе имеется необъявленный идентификатор. И даже подсказано какой: идентификатор ***c***. Чтобы отыскать строку с ошибкой, достаточно дважды кликнуть мышкой по строке с описанием ошибки. В тексте программы возле строки с ошибкой появится знак указателя.

Итак, ошибка в операторе ***Console.WriteLine(c);*** Именно здесь имеется идентификатор ***c***, который является переменной. Значение этой переменной мы пытаемся вывести на экран. Но по правилам языка можно использовать только те идентификаторы, которые были ранее объявлены. Среди объявленных переменных нет такой, но есть переменная ***c1***, в которой на самом деле и находится результат вычисления. Значит, или объявлена не та переменная, или в использовании ошибочно описан другой идентификатор. Что-то надо исправить. В данном случае можно править и так, и так, но проще, конечно, исправить имя в операторе вывода.

Сделаем, например, так. Исправим идентификатор: ***Console.WriteLine(c1);*** и вновь выполним компиляцию. Теперь ошибок нет. Можно попытаться выполнить программу.

Выполнение программы.



Для выполнения программы нажмем комбинацию клавиш ***Ctrl-F5.*** Получим черное окно (слева). Это окно, в котором отображаются результаты выполнения нашей программы. Так как наша программа сделана в виде консольного приложения, то мы видим на экране окно DOS-приложения. Виден результат выполнения оператора вывода и строка, которая предлагает для окончательного завершения работы программы нажать любую клавишу.

Если бы мы запустили программу, выбрав пункт меню ***Отладка – Начать отладку*** или нажав соответствующую кнопку

на панели инструментов, то этой строки не было бы. Проверим это. Черное окно мелькнет, и исчезнет (закроется) без всяких сообщений. Так и есть: в тексте программы нет никаких задержек. Для того, чтобы окно задержалось, надо в конце программы добавить оператор `Console.ReadKey();` или `Console.ReadLine();`

Откроем папку ***proba\proba\bin\debug*** (окно слева). В этой папке размещена готовая программа ***Proba*** (EXE-файл). Запустите его на выполнение. Посмотрите результат. Нажмите любую клавишу. Работа с программой завершена. Завершите работу с программной средой.

Выводы и замечания.

1. Мы рассмотрели только основные возможности работы со средой Visual Studio в части использования языка C#. Работая аналогично, вы сможете создавать свои решения и проекты и свои программы.

Индивидуальное Задание 1.

1. Разработайте собственный простой проект по своему варианту по заданию 1 из лабораторной работы 1. Значения входных переменных задавайте с помощью оператора присваивания (оператор ввода пока не используйте).

Задание 2

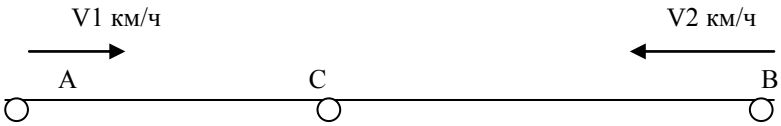
Прочитайте и выполните следующее задание:

Условие задачи.

Из пункта А в пункт В выехали навстречу друг другу два велосипедиста. Скорость первого – V1 км/ч, скорость второго – V2 км/ч. Найти путь, который преодолеет каждый велосипедист до встречи, если расстояние между А и В известно.

Анализ условия задачи.

Для нахождения алгоритма решения задачи построим чертеж.



Точка С – это место встречи. Очевидно, что до момента встречи каждый велосипедист будет в пути одинаковое время. Пусть это будет время Т. Тогда путь, который преодолеет первый велосипедист, вычисляется по формуле: S1=V1 * T. Для второго велосипедиста справедлива такая формула: S2=V2 * T. Если весь путь обозначить как S, то S=S1+S2 или

S=V1*T+V2*T=T*(V1+V2). Следовательно:

$$T = \frac{S}{V2 + V1}$$

Тогда, S1 и S2 можно легко рассчитать по ранее приведенным формулам:

$$S1 = V1 * T$$
$$S2 = V2 * T$$

Исполнение программы.

Попробуем для решения задачи использовать следующий текст:

```
double V1, V2, S, S1, S2, T;
Console.WriteLine("Скорость первого=");
V1 = Console.ReadLine();
Console.WriteLine("Скорость второго=");
V2 = Console.ReadLine();
Console.WriteLine("Расстояние=");
S = Console.ReadLine();
T = S / (V1 - V2);
S1 = V1 * T;
S2 = V2 * T;
Console.WriteLine("Путь первого= " + S1);
```

```
Console.WriteLine("Путь второго= " + S2);
```

Создайте проект и скопируйте текст этот в вашу программу. Давайте внимательно рассмотрим текст. В программе объявляются шесть вещественных переменных удвоенной точности. Это понятно: мы будем работать с числами. С клавиатуры вводятся три значения – две скорости и расстояние. Затем производится вычисление общего времени и пройденных путей. Полученные значения выводятся на экран. Как будто бы все верно и логично. Но как это смотрится с точки зрения синтаксиса языка C#? Выполним компиляцию.

Увы, по итогам компиляции мы получили три ошибки. Все с одинаковым сообщением***: Неявное преобразование типа "string" в "double" невозможно.*** При дословном переводе имеем: «*не могу преобразовать тип строка в тип вещественный*». Надо сказать, что подобные сообщения характерны для программирования в среде C#. Язык очень чувствителен к правильности использования **типов данных**. В нашем случае мы очень вольно поступили с методом ***ReadLine***. Он позволяет вводить с клавиатуры **строки**, а мы пытаемся присвоить введенную строку **вещественной** переменной. Значит, после ввода строки ее надо преобразовать в вещественный тип и лишь потом присваивать.

Поступим так. Объявим в программе строковую переменную ***temp*** и будем в нее вводить данные с клавиатуры. Сразу после ввода сделаем преобразование в число и выполним присваивание. Получим такой текст:

```
double V1, V2, S, S1, S2, T;
string temp;                                    // добавлено
Console.WriteLine("Скорость первого=");
temp = Console.ReadLine();                      // изменено
V1 = Convert.ToDouble(temp);                    // добавлено
Console.WriteLine("Скорость второго=");
temp = Console.ReadLine();                      // изменено
V2 = Convert.ToDouble(temp);                    // добавлено
Console.WriteLine("Расстояние=");
temp = Console.ReadLine();                      // изменено
S = Convert.ToDouble(temp);                     // добавлено
T = S / (V1 - V2);
S1 = V1 * T;
S2 = V2 * T;
Console.WriteLine("Путь первого= " + S1);
Console.WriteLine("Путь второго= " + S2);
```

Выполним компиляцию и увидим, что больше синтаксических ошибок нет:

```
namespace proba
{
    class Program
```

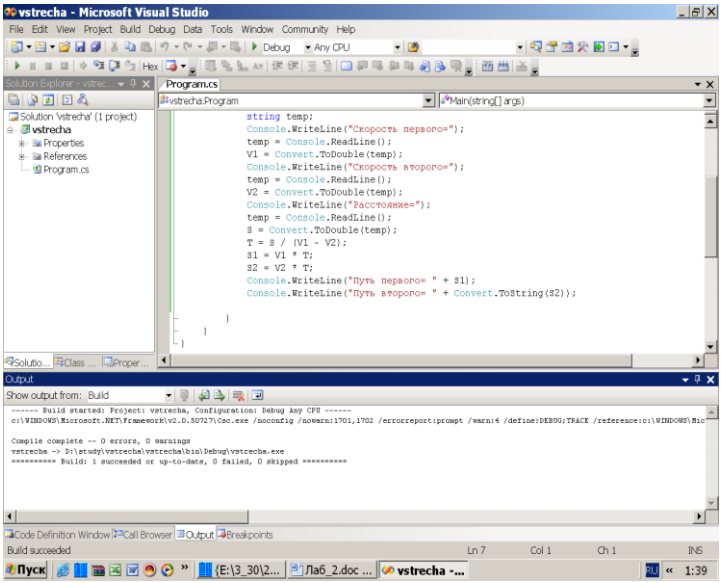
```
    {
        static void Main(string[] args)
        {
            double V1, V2, S, S1, S2, T;
            string temp;                            // добавлено
            Console.WriteLine("Скорость первого=");
            temp = Console.ReadLine();              // изменено
            V1 = Convert.ToDouble(temp);            // добавлено
            Console.WriteLine("Скорость второго=");
            temp = Console.ReadLine();              // изменено
            V2 = Convert.ToDouble(temp);            // добавлено
            Console.WriteLine("Расстояние=");
            temp = Console.ReadLine();              // изменено
            S = Convert.ToDouble(temp);             // добавлено
            T = S / (V1 - V2);
            S1 = V1 * T;
            S2 = V2 * T;
            Console.WriteLine("Путь первого= " + S1);
            Console.WriteLine("Путь второго= " + S2);
        }
    }
}
```

Все хорошо, но почему же компилятор не обнаружил ошибок в двух последних операторах? Ведь метод ***WriteLine*** также работает со строками, а переменные S1 и S2 – вещественные! Более того, в этих операторах мы используем операцию со знаком + (**конкатенация** строк) для объединения в одно целое строки (в двойных кавычках) и вещественного числа – и тоже нет ошибки!

Дело в том, что компилятор при выполнении объединения **неявно** преобразует вещественное число в строку, а уж потом объединяет их. Число преобразуется именно в строку, потому что первый операнд в этой операции – строка. Такие неявные преобразования среда выполняет довольно часто. Она как бы подправляет неточности программиста, но подправляет их по своему пониманию. Если уж быть точным, то следовао бы явно описать преобразование и не надеяться на компилятор – а вдруг он сделает не то, что нужно? Явное преобразование можно было бы сделать используя метод ***ToString*** класса ***Convert.*** Например:

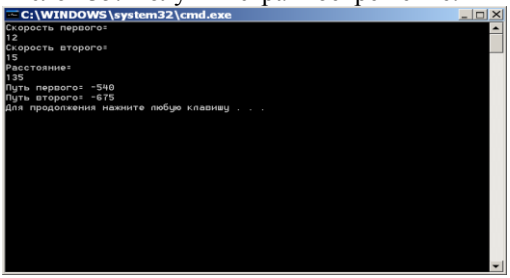
Console.WriteLine("Путь второго= "+Convert.ToString(S2));

Поправьте (ради разнообразия) второй оператор вывода результатов и вновь скомпилируйте программу. Убедитесь, что синтаксических ошибок нет.



Теперь можно выполнить программу. Нажимаем ***CTRL-F5***.

Введем в качестве скоростей числа 12 и 15, а в качестве расстояния – число 135. Получим странное решение.

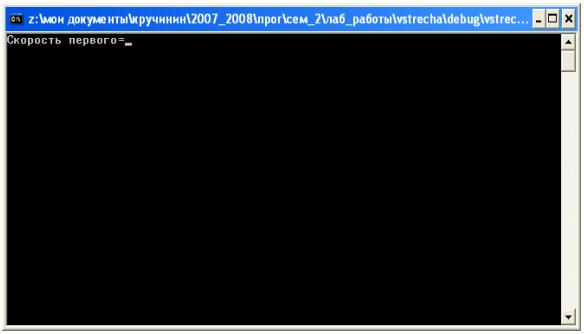


Величина пройденного пути не может быть отрицательным числом. От синтаксических ошибок мы избавились, но где-то в вычислениях есть логическая ошибка. Завершим исполнение программы, нажав любую клавишу.

Проведите расчет вручную по формулам. Путем ручного подсчета можно убедиться, что должны получиться значения 60 и 75. Где же ошибка? Выполним нашу программу в **режиме отладки**. Суть режима в том, что система при выполнении программы дает программисту возможность следить, как производятся вычисления на любом шаге, при выполнении любого оператора. При этом можно пропускать в автоматическом режиме те участки программы, которые вне подозрения и задерживаться на тех, где может быть ошибка.

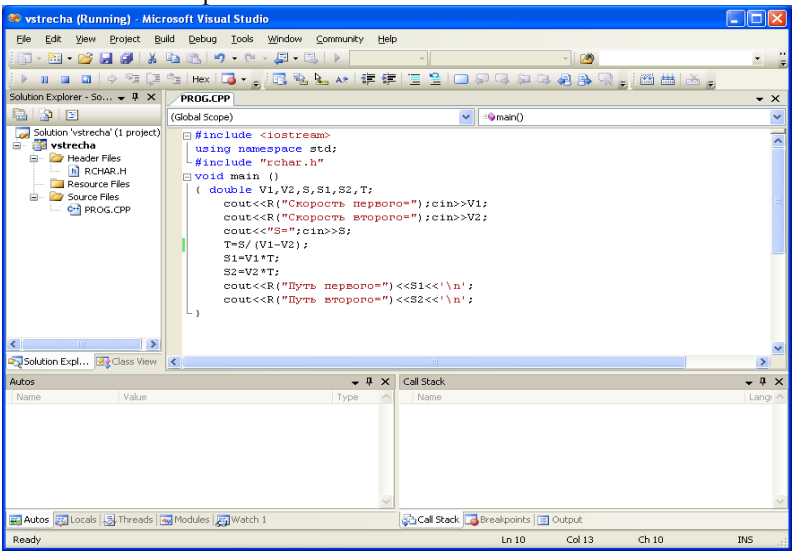
Итак, Наша задача – увидеть именно **неправильность вычисления, проведя выполнение программы по шагам.** Попробуем.

Выполнение программы в режиме отладки.

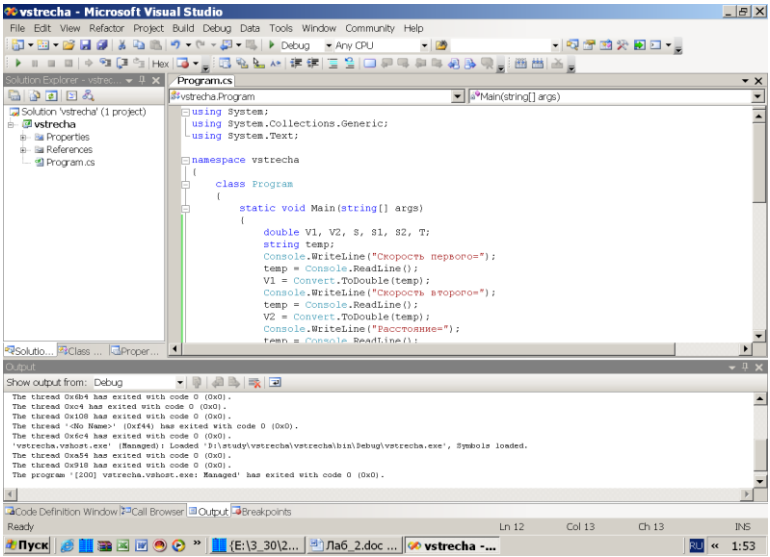


Обычно мы запускаем программу, используя комбинацию клавиш ***CTRL-F5.*** Для выполнения в режиме отладки достаточно нажать просто ***F5***. Нажимаем. Мы увидим следующее окно (слева).

Появится оно на фоне такого окна:

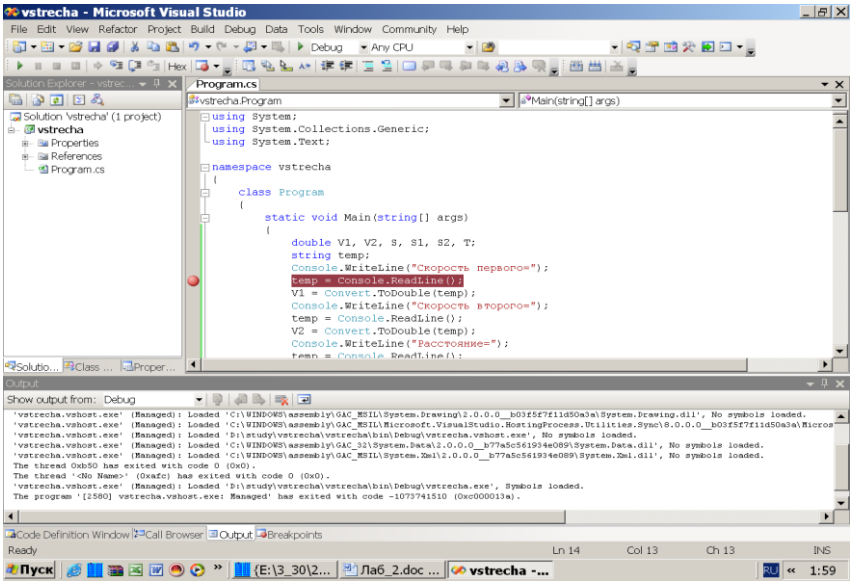


Введем в черном окне значения 12, 15 и 135. Что-то очень быстро просчитается, и в итоге получится окно:

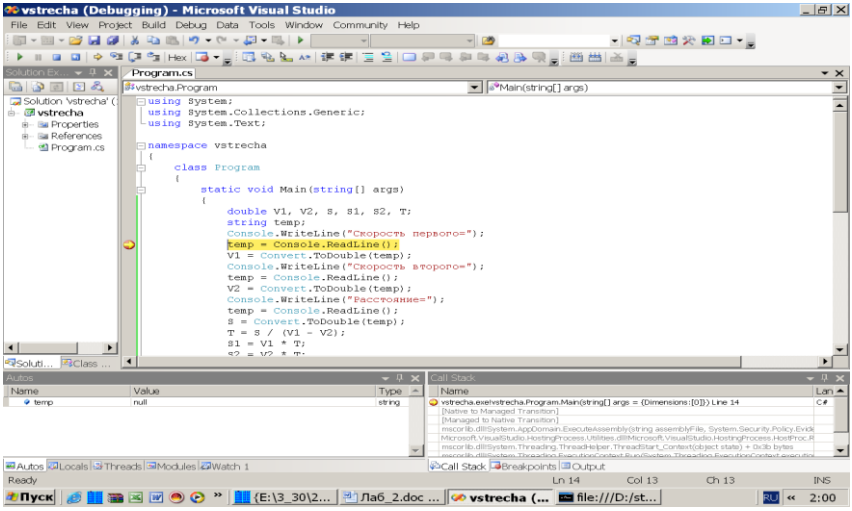


В окне вывода что-то появилось, что-то мелькнуло, но мы ничего не смогли отследить. Более того, черное окно вообще исчезло, и мы не увидели никаких результатов работы. При запуске по ***CTRL-F5*** мы хотя бы увидели неправильные результаты, а здесь – вообще ничего! Все дело в том, что при выполнении отладки черное окно закрывается автоматически – так же, как при запуске программы из готового файла типа ***exe*** (это мы видели в предыдущей лабораторной работе). Чтобы можно было следить за выполнением работы программы, нам следует сначала выполнить некоторые подготовительные действия. Мы можем указать, что процесс исполнения программы должен выполняться автоматически, но не от начала до конца, а от начала до некоторого оператора. Пусть таким оператором будет, например, самый первый из операторов вывода: ***`temp = Console.ReadLine();`***

Для того, чтобы программа остановилась перед этим оператором (**до начала** выполнения этого оператора), отметим этот оператор специальным значком – знаком «**контрольная точка**». Сделать это несложно. Щелкнем мышкой в строке этого оператора на сером фоне.

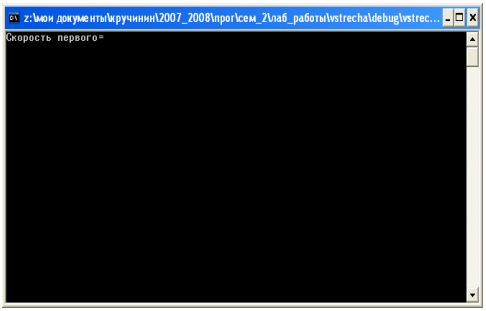


Слева от оператора увидим значок, который и означает, что контрольная точка установлена. Теперь нажимаем ***F5.*** Опять появляются два окна. Одно – это черное окно исполнения программы, а другое такого вида:

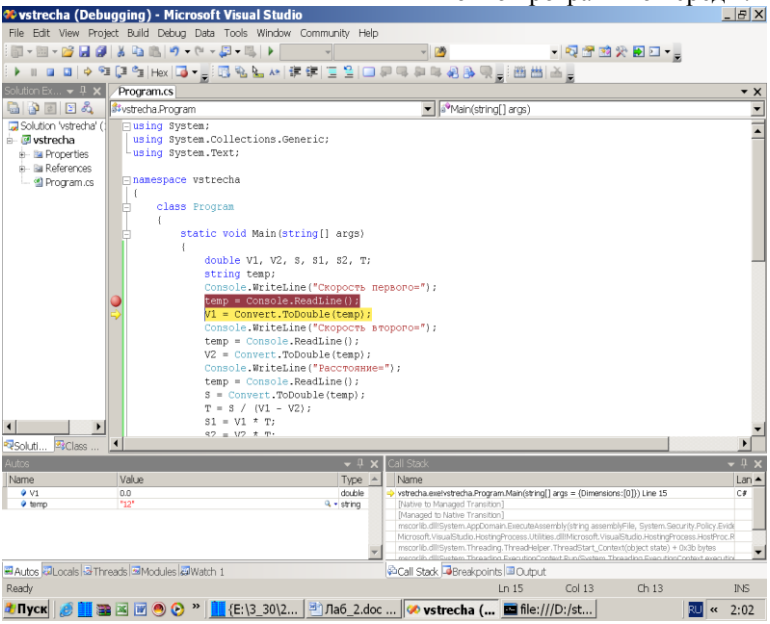


Обратите внимание на желтую стрелочку возле указанного оператора. Эта стрелка означает, что выполнение программы остановлено **до**

начала выполнения этого оператора.



Выполним оператор. Для этого нажмем ***F10.*** Произойдет переключение в черное окно. Сообщение «Скорость первого = » появилось. Программа перешла к выполнению *ReadLine*. Программа ждет нашего ответа. Введем число 12 и нажмем ***Enter.*** Произойдет переключение в окно программной среды:

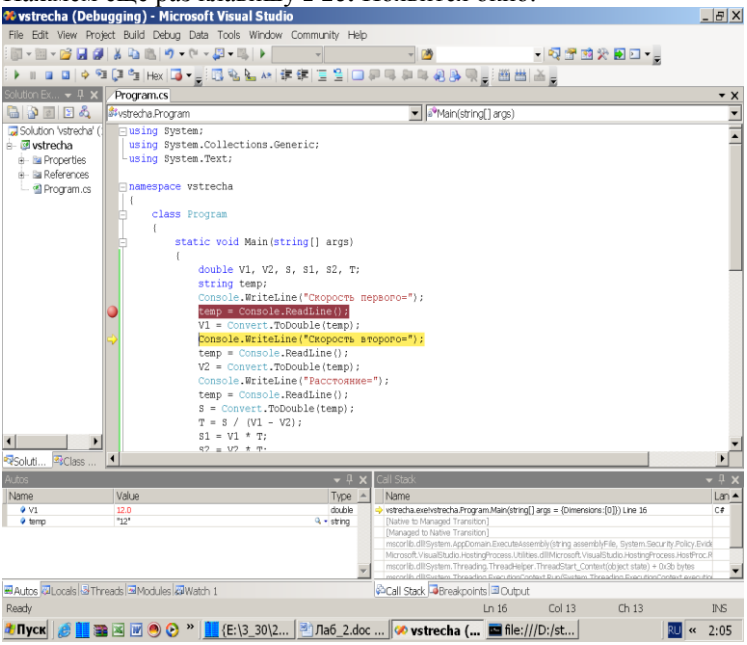


Что в нем необычного?

Во-первых, желтая стрелка переместилась на следующую строку. Это значит, что программа вновь остановилась перед следующим оператором. Нажав ранее ***F10,*** мы дали команду «выполнить один шаг программы».

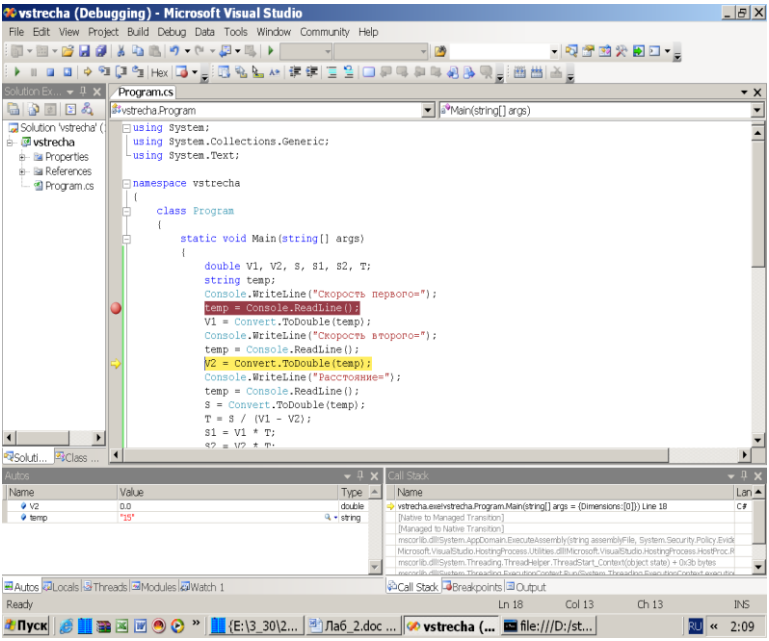
Во-вторых, результаты выполнения этого шага отразились в левом нижнем окне. В нем можно увидеть, какие значения имеют переменные, используемые в нашей программе. Например, переменная ***V1*** имеет значение 0, а переменная ***temp*** имеет значение 12 – мы его только что ввели, о чем нам подсказано красным цветом.

Нажмем еще раз клавишу ***F10.*** Появится окно:

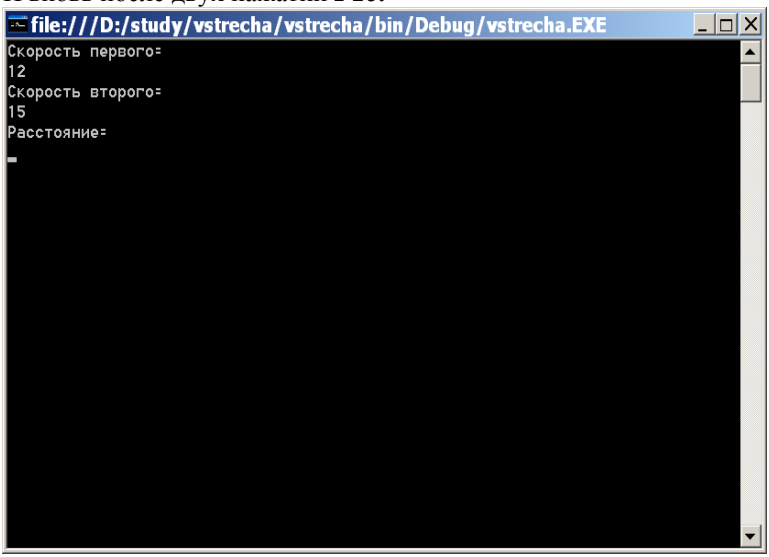


Мы видим, что переменная V1 получила вещественное значение. Нажимая F10 два раза подряд, приходим к окну:

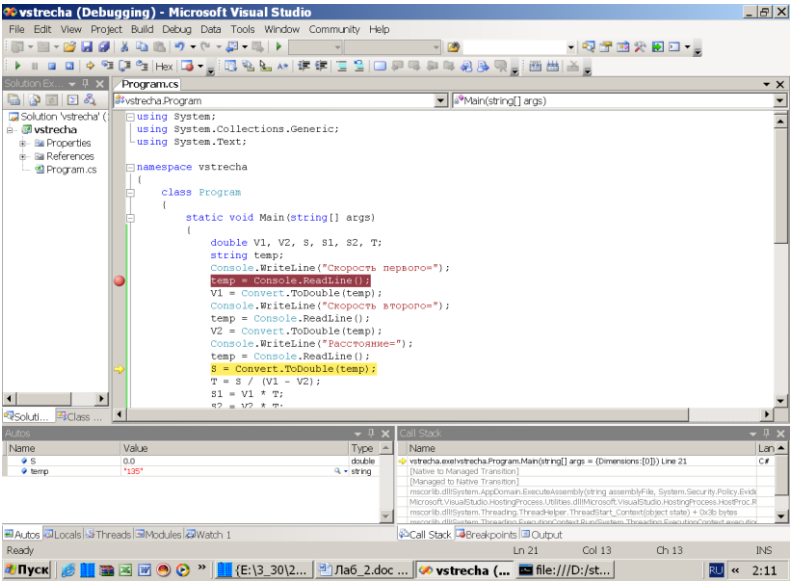
Вводим следующее значение – 15.



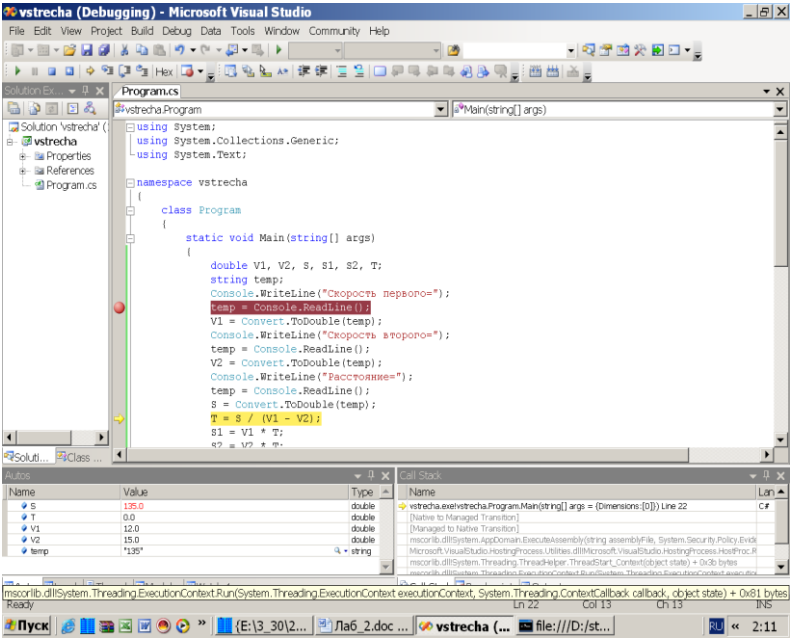
И вновь после двух нажатий ***F10.***



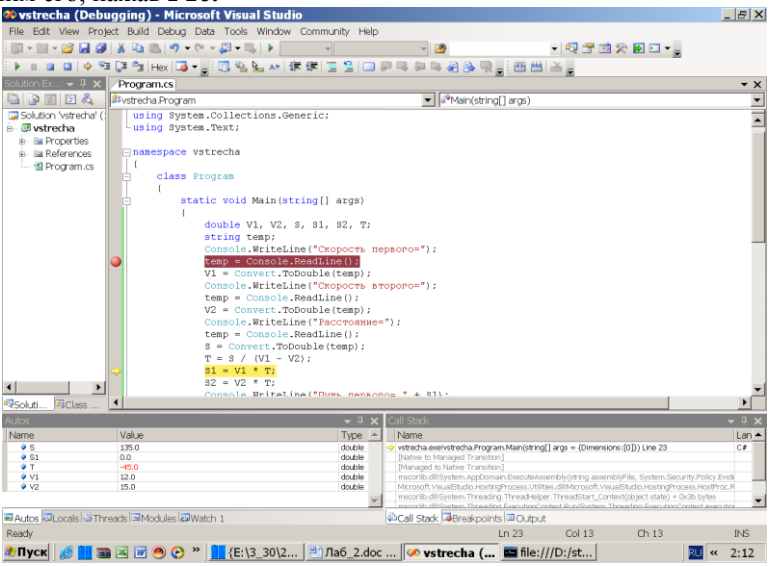
Вводим значение расстояния – 135.


vstrecha (Debugging) - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Window Community Help
Program.cs
vstrecha.Program
Main(string[] args)
using System;
using System.Collections.Generic;
using System.Text;
namespace vstrecha
{
class Program
{
static void Main(string[] args)
{
double V1, V2, S, S1, S2, T;
string temp;
Console.WriteLine("Скорость первого=");
temp = Console.ReadLine();
V1 = Convert.ToDouble(temp);
Console.WriteLine("Скорость второго=");
temp = Console.ReadLine();
V2 = Convert.ToDouble(temp);
Console.WriteLine("Расстояние=");
temp = Console.ReadLine();
S = Convert.ToDouble(temp);
T = S / (V1 - V2);
S1 = V1 * T;
Solution 'vstrecha' (
vstrecha
Properties
References
Program.cs
Name
Value
Type
S
0.0
double
temp
"135"
string
Call Stack
Name
vstrecha.exe!vstrecha.Program.Main(string[] args = {Dimensions:[0]}) Line 21
[Native to Managed Transition]
[Managed to Native Transition]
Autos
Locals
Threads
Modules
Watch 1
Call Stack
Breakpoints
Output
Ready
Ln 21
Col 13
Ch 13
INS
Пуск
{E:\3_30\2...
Лаб_2.doc ...
vstrecha (...
file:///D:/st...
2:11

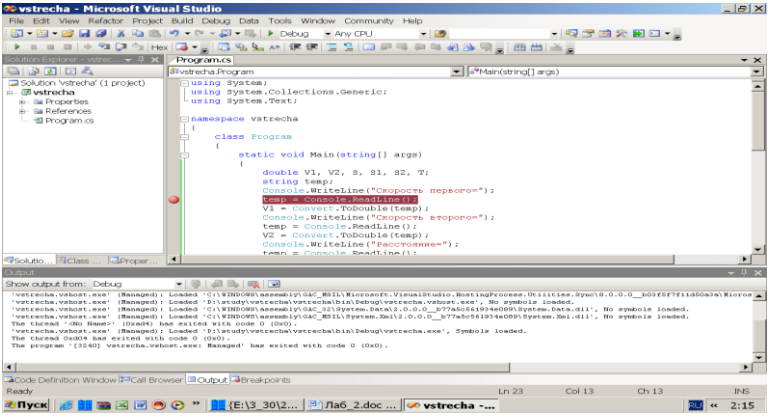
Снова F10.


vstrecha (Debugging) - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Window Community Help
Program.cs
vstrecha.Program
Main(string[] args)
using System;
using System.Collections.Generic;
using System.Text;
namespace vstrecha
{
class Program
{
static void Main(string[] args)
{
double V1, V2, S, S1, S2, T;
string temp;
Console.WriteLine("Скорость первого=");
temp = Console.ReadLine();
V1 = Convert.ToDouble(temp);
Console.WriteLine("Скорость второго=");
temp = Console.ReadLine();
V2 = Convert.ToDouble(temp);
Console.WriteLine("Расстояние=");
temp = Console.ReadLine();
S = Convert.ToDouble(temp);
T = S / (V1 - V2);
S1 = V1 * T;
Solution 'vstrecha' (
vstrecha
Properties
References
Program.cs
Name
Value
Type
S
135.0
double
T
0.0
double
V1
12.0
double
V2
15.0
double
temp
"135"
string
Call Stack
Name
vstrecha.exe!vstrecha.Program.Main(string[] args = {Dimensions:[0]}) Line 22
[Native to Managed Transition]
[Managed to Native Transition]
mscorlib.dll!System.Threading.ExecutionContext.Run(System.Threading.ExecutionContext executionContext, System.Threading.ContextCallback callback, object state) + 0x81 bytes
Ready
Ln 22
Col 13
Ch 13
INS
Пуск
{E:\3_30\2...
Лаб_2.doc ...
vstrecha (...
file:///D:/st...
2:11

Обратите внимание: уже введены значения переменных ***V1, V2*** и ***S.*** Стрелка остановилась перед оператором вычисления переменной ***T.*** Выполним его, нажав ***F10.***

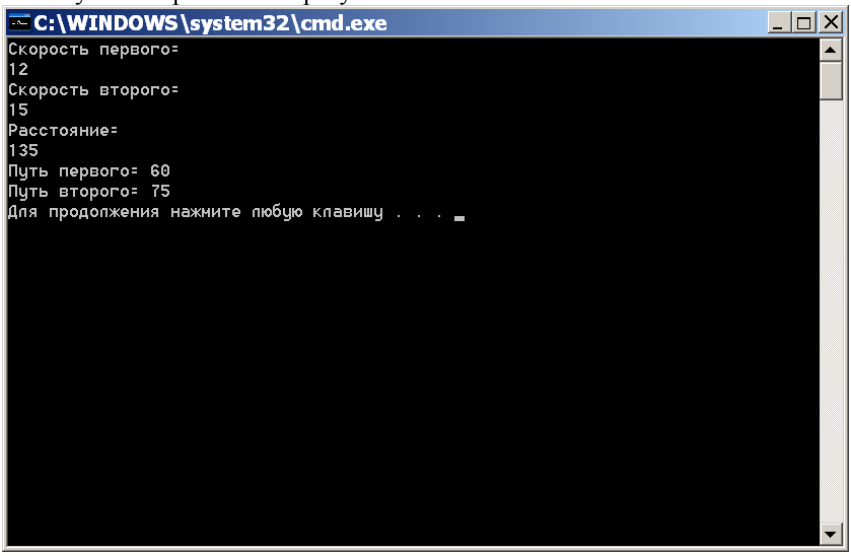


А вот теперь есть смысл насторожиться. Время получилось **отрицательное**! Мы или жить начинаем вспять, или … в этом операторе есть ошибка. Первое невозможно, а второе – имеет место быть. Надо V1+V2, а у нас V1 – V2. Теперь можно прервать процесс отладки, нажав ***SHIFT-F5.***



Снимем знак контрольной точки, щелкнул мышкой на сером фоне. Исправим знак «минус» на знак «плюс» и выполним программу без от-

ладки. Получится правильный результат.



Нажмем любую клавишу для завершения исполнения программы.

***Выводы и замечания*.**

1. Не всегда только что написанный текст программы безошибочен – при написании возможны **синтаксические** ошибки, которые обнаруживаются компилятором при отладке. Очень часто такие ошибки бывают связаны с неправильным использованием типов данных.
2. Не всегда синтаксически правильный текст правильно работает. Могут быть **логические** ошибки. Если их не удается обнаружить визуально, то можно выполнить программу в режиме отладки.
3. В процессе отладки можно устанавливать любое количество контрольных точек.

Индивидуальное задание 2.

Выполнить задание 2 из лабораторной работы №1 (по своему варианту). Использовать режим отладки, установив несколько контрольных точек.

Указание:

Язык C# имеет большую коллекцию функций математической обработки данных, которые реализованы в классе **Math** пространства имен

System. Основные из них приведены в таблице:

Запись на C#	Возвращаемый результат
Math.Abs(X);	Модуль числа X
Math.Ceiling (X);	Округление числа X до большего целого
Math.Floor(X);	Округление числа X до меньшего целого
Math.Cos (X);	Косинус аргумента X
Math.E	Число e. e = 2,718282
Math.Exp (X);	Экспонента, число e в степени X
Math.Log(X);	Логарифм натуральный числа X
Math.Log10(X);	Логарифм десятичный числа X
Math.Max(X,Y);	Максимум из двух чисел X и Y.
Math.Min (X,Y);	Минимум из двух чисел X и Y.
Math.Pi	Число пи.
Math.Pow(X,Y);	Число X в степени Y
Math.Round(X);	Простое округление числа X
Math.Sing(X);	Знак числа X
Math.Sin(X);	Синус аргумента X
Math.Sqrt(X);	Квадратный корень числа X
Math.Tan(X);	Тангенс аргумента X

Примечание: аргументы тригонометрических функций задаются в радианах.