## Тема 2

## Основные средства языка программирования VB .NET. Типы данных

### 4.2.1. Синтаксис, семантика, алфавит и лексемы языка VB

Содержательно язык программирования – это средство общения между человеком (разработчиком программ) и компьютером (исполнителем программ). Рассматривая любую знаковую систему (в том числе и язык программирования), обычно выделяют ***синтаксис*** – правила построения сообщений в этой системе, ***семантику*** – правила истолкования сообщений тем, кому они адресованы, а также ***прагматику*,** сопоставляющую сообщения желаниям того, от кого они исходят.

Всякий язык программирования, в частности, VB, можно определить, как множество предложений (строк) – т.е. некоторое множество конечных последовательностей элементарных единиц из некоторого непустого конечного множества символов (алфавита), называемого словарем языка. Понятно, что при таком рассмотрении языка программирования мы только фиксируем множество символов, которые можно использовать для записи программ, а также класс допустимых (или, как принято говорить, синтаксически правильных) программ, не сопоставляя никакого смысла этим синтаксически правильным программам. Ясно, что задавать множество допустимых программ исчерпывающим их перечислением невозможно. Желательно, чтобы описание языка было обозримым (заведомо конечным). Обычный подход, удовлетворяющий этому требованию, состоит в том, что предложения (строки) языка строятся по определенным правилам, в совокупности составляющим то, что называют *грамматикой* языка. Эти грамматические правила приписывают предложениям языка некоторую синтаксическую структуру, которая используется при определении смысла предложений.

***Грамматика*** языка может быть описана различными способами. Например, ее можно задавать в виде порождающей системы формальных и неформальных правил, т.е. набора правил, применением которых можно породить все предложения языка и только их. При описании конструкций языка VB мы будем применять способ, использующий неформальные правила описания предложений языка VB.

***Семантика*** языка программирования – это правила придания смысла синтаксически правильным программам. В конечном счете, эти правила определяют ту последовательность действий вычислительной машины, которую она должна выполнить, работая по данной программе. Например, семантика языка команд компьютера определяется самим компьютером: машинная программа описывает в точности то, что реализует вычислительная машина при работе по данной программе. Аналогично семантика языка программирования высокого уровня может быть определена через описание правил выполнения соответствующей виртуальной машины.

***Алфавит языка*** использует для записи программ сравнительно небольшое множество литер (символов), доступных на устройствах ввода-вывода реальных ПК. Предполагается, что ***входной алфавит*** содержит буквы русского и латинского алфавитов, десятичные арабские цифры и некоторое количество специальных ***литер,*** таких как пробел, точка, запятая и т.п.

Таким образом, ***в алфавит языка программирования* VB** входят:

* прописные (**A – Z**) и строчные (**a – z**) буквы латинского алфавита;
* прописные (**А – Я**) и строчные **(а – я**) буквы кириллицы;
* цифры от **0** до **9**;
* символ подчеркивания – \_;
* неотображаемые символы (пробел, табуляция, переход на новую строку);
* специальные символы, участвующие в построении конструкций языка: +- \*/; \^ =><[ ] () . :{} ’ ”& @;
* составные символы, воспринимаемые как один символ: <=, >=, <>.

Для записи даже элементарных конструкций языка VB используются не отдельные ***литеры (символы)*** входного алфавита, а ***слова***, называемые ***лексемами***. При этом символы, образующие лексему, теряют свою индивидуальность: ***смысловое значение, приписываемое лексеме семантикой, не обязано вытекать из смысловых значений составляющих ее символов.***

Множество **лексем** образует **словарь языка**. Он включает в себя следующие четыре группы лексем:

* множество **специальных символов**, состоящее из знаков операций (например, +, \*) и **ограничителей** (например, точка, запятая);
* множество зарезервированных (ключевых) слов (например, **Sub, End, If**);
* множество **идентификаторов**, используемых в качестве имен констант, переменных, типов, объектов, модулей, процедур и функций и т.д.;
* множество изображений **константных значений**, состоящее из целых и вещественных чисел и символьных и строковых констант и т.д.

Рассмотрим правила использования некоторых лексических элементов.

***Идентификаторы***– это имена лексического элемента языка программирования. Для выбора имен и лексического элемента языка следует придерживаться следующих правил:

* имя должно состоять только из символов латинского алфавита или кириллицы (регистр значения не имеет), цифр и подчеркивания;
* имя должно начинаться с буквы;
* имена не могут совпадать с зарезервированными словами;
* имена не должны содержать более 255 символов;
* не следует использовать имена, совпадающие с названиями функций, и методов языка;
* нельзя использовать одни и те же имена переменных, отличающихся типом;
* не допускается использование повторяющихся имен на одном уровне области определения.

***Ключевые слова***– это идентификаторы, зарезервированные для специального использования (например, **Const, Do, For, If, While**) и не использующиеся как имена для программных элементов, таких, например, как переменные и процедуры.

***Операторы языка программирования*** – это последовательности команд, из которых состоит программный код (текст программы). В то же время ***оператор***– это синтаксическая единица языка програм­мирования, которая определяет некоторое действие. Операторы описывают алгорит­мические действия, которые должны выполняться. Каждый оператор представляет собой законченную фразу (предложение, строку) языка программирования и может содержать ключевые слова, переменные, константы, выражения, разделители др. Операторы могут быть простыми и структурированными. Простым опера­тором является такой оператор, который не содержит в себе других операторов. Структурированные операторы строятся из других операторов, порядок выполне­ния которых должен быть последовательным, определяемым условной передачей управления (условными операторами) или повторяющимся (определяется операторами цикла). Обычно оператор располагается на одной строке, но его можно продолжить на другой строке с помощью символа продолжения строки – знака – ''***\_***''*(подчеркивание), причем символу продолжения должен предшествовать пробел; или на одной строке можно записать и несколько операторов, разделяя их знаком* – *: (двоеточие).*

***Комментарии***– это пояснения отдельных конструкций и их действий в программе. *Строки комментария начинаются со знака –' (апостроф).* ***Комментарии*** обычно вставляют в начале или в конце строки программного кода. При выполнении программы ***комментарии*** игнорируется. В программном коде по умолчанию комментарий выделя­ется зеленым цветом. В нашем пособии в отдельных фрагментах, процедурах и проектах комментарии будем писать курсивом.

***Данные***– это информация, представленная в формальном виде, который обес­печивает возможность ее хранения, обработки и передачи. В основе любого языка программирования лежит конечное множество данных, неделимых (простых) с точки зрения имеющихся в языке операций. Это множество распадается на четыре подмножества, состоящих соответственно из целых чисел, вещественных чисел, логических значений и литерных (строковых) значений. Каждое из перечисленных подмножеств – это ***стандартный тип данных***, ***который имеет свое имя (идентификатор), характеризуется множеством значений и набором операций***, определяемых языком программирования над этими значениями (например, числа можно складывать, а логические или литерные значения – нет). Как известно, данные в программе могут быть двух видов: постоянные величины (***константы***) и пере­менные величины (***переменные***). Предполагается, что каждая переменная в программе (переменная – это элемент памяти ПК; адрес элемента памяти – это имя переменной; а его содержимое – значение переменной) может получать значения только определенного заранее типа, в то время как элементы памяти могут хранить любое по типу значение – его трактовка определяется только выполняемой операцией. Помимо переменных программа может содержать константы – элементы памяти ПК, которые постоянно хранят заданные значения. В VB значение константы может использоваться двумя способами: или неявно, своим именем – адресом соответствующего элемента памяти, или явно, изображением значения содержимого соответствующего элемента памяти. Например, **True**– это имя значения «***истина***», а 25 – изображение значения числа «***двадцать пять***».

### 4.2.2. Типы данных, способы их объявления, переменные, константы и оператор присваивания

Итак, переменные используются во время выполнения программы для временного хранения данных в памяти компьютера. Для ***определения (объявления)*** переменных компилятору нужно следующее:

* ***имя переменной*** – по имени компилятор осуществляет связь переменной в программе с областью оперативной памяти компьютера;
* ***тип переменной***– тип позволяет компилятору определить, какого вида информация хранится в переменной;
* ***значение переменной***– определяет содержание информации, которая хранится в памяти компьютера.

При выборе имен переменных следует исполь­зовать имена, которые являются короткими и, в то же время, интуитивно понят­ными и простыми для запоминания. Чтобы избежать противоречий, рекомендуется использовать при создании имен переменных следующие соглашения:

* начинать имя каждой переменной с буквы или знака подчеркивания. (это тре­бование VB – имена переменных могут содержать только буквы, цифры и знак подчеркивания);
* выбирать имена переменных не длиннее 33 символов;
* делать имена своих переменных описательными, комбинируя, когда это имеет смысл, одно или несколько слов;
* использовать комбинации заглавных и строчных символов и цифр, то есть первую букву каждого слова имени переменной делать заглавной;
* не использовать в качестве имен переменных ключевые слова VB, име­на объектов или свойств (если вы это сделаете, то при попытке запуска про­граммы получите сообщение об ошибке).

Какими бы типами объектов, мы не пользовались при написании программы, следует помнить, что основой их являются простейшие (базовые) типы. Это основные элементы информации, из которых будет формироваться впоследствии сложная структура объектов. Следует точно знать, какой именно вид информации хранится в той или иной переменной, и в соответствии с этим знанием применять к ней те или иные операции или функции обработки. Обратите внимание, что VB имеет достаточно хорошую типизацию данных.

Таблица 4.2.2-1 содержит перечень базовых (простых) ти­пов данных VB. Причем каждый базовый тип языка VB соответствует типу данных .NET Framework.

*Таблица 4.2.2-1*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Тип**  **данных** | **бит** | **Диапазон** | **Пример использования** | **Что хранит**  **(Тип .NET)** |
| **Short** | *16* | от-32768  до 32767 | **Dim Birds As Short**  **Birds = 12500** | Целые числа  (System. Int) |
| **UShort** | *16* | от 0  до 65535 | **DimDays As UShort**  **Days = 55000** | Целые числа  (System.UInt) |
| **Integer** | *32* | от-2147483648  до 2147483647 | **DimInsects As Integer**  **Insects = 37500000** | Целые числа  (System.Int32) |
| **UInteger** | *32* | от 0  до 4294967295 | **DimJoys As UInteger**  **Joys = 3000000000** | Целые числа  (System.UInt32) |
| **Long** | *64* | от-9223372036854775808  до 9223372036854775807 | **DimWorldPopAs Long**  **WorldPop = 4800000004** | Целые числа  (System.Int64) |
| **ULong** | *64* | от 0  до 18446744073709551615 | **Dim Stars As ULong**  **Stars=**  **180000000000000000** | Целые числа  (System.UInt64) |
| **Single** | *32* | от -3.4028235Е38  до 3.4028235Е38 | **DimPrice As Single**  **Price = 899.99** | Числа с пл. точкой  (System.Single) |
| **Double** | *64* | от -1.79769313486231Е308  до 1.79769313486231Е308 | **DimPi As Double**  **Pi = 3.1415926535** | Числа с пл. точкой  (System.Double) |
| **Decimal** | *128* | Полож. и отриц. значения  до 29 знаков | **DimDebt As Decimal**  **Debt =7600300.50** | Числа с фик. Точкой (System.Decimal) |
| **Byte** | *8* | от 0  до 255 без знака | **DimRetKeyAs Byte**  **RetKey = 13** | Десятичные числа  (System.Byte) |
| **SByte** | *8* | от-128  до127 | **DimNegValAsSByte**  **NegVal = -20** | Целые числа  (System.SByte) |
| **Char** | *16* | любой символ Unicode в диапазоне 0 - 65535 | **DimUCharAsCharUChar="A"** | Один символ  (System.Char) |
| **String** | *16* | от0 до ≈ 2 миллиардов символов Unicode | **Dim Dog As String**  **Dog ="pointer"** | Текст  (System.String) |
| **Boolean** | *16* | True или False (при преобразовании типов 0 преобразуется в False, а  все другие значения в True) | **DimFlag As Boolean**  **Flag = True** | Логические знач.  (System.Boolean) |
| **Data** | *64* | от 1 января 0001  до 31 декабря 9999 | **DimBirthday As Date**  **Birthday=#3/1/1963#** | Дату и время  (System.DateTime) |
| **Object** | *32* | любой тип, который может быть сохранен в  переменной типа Object | **DimApp As Object**  **App = CreateObject \_ ("Word.Application")** | Ссылка на объект  (System.Object) |

В программном коде может использоваться любое количество переменных, и они могут со­держать слова, числа, даты, свойства или другие значения. Использова­ние переменных позволяет присвоить каждой части данных, с которыми вы работаете, короткие и легко запоминаемые имена. Переменные могут хранить информацию, введенную пользователем при выполнении про­граммы, результат некоторого вычисления или данные, которые отображаются на форме. Если коротко, то переменные явля­ются удобными ***контейнерами****,* которые используются для хранения ин­формации почти любого типа.

Как было отмечено ранее, использование переменных в программе на VBтребует не­которого планирования. Прежде чем использовать переменную, необ­ходимо выделить в компьютере память, которую эта переменная будет занимать. Этот процесс несколько напоминает бронирование мест в театре или на спортивном матче.

Для каждого типа переменных выделяется определенное количество оперативной памяти, и тип данных надо выбирать, исходя из задания на решение конкретной задачи. Если данные должны принимать только целые значения, то тип соответствующих переменных в программе должен быть **Integer,** но никак не **Single**, поскольку в этом случае мы увеличиваем объем используемой оперативной памяти и не гарантируем получение правильного (точного) результата. Если из постановки задачи известно, что данные могут принимать дробные значения, то соответствующим переменным в программе необходимо назначить тип **Single** или **Double**, поскольку любой другой тип переменных может привести к непредвиденным результатам. Поэтому в VB следует ***всегда явно объявлять переменные до их использования.*** А для того чтобы VB информировал программистов о том, что переменная явно не объявлена до ее использования, ***необходимо в начале программного кода (перед всеми процедурами) и объявлениями поместить оператор:* Option Explicit Оn.**

Чтобы в VB ***явно объявить*** переменную, необходимо ввести ***имя пере­менной*** после оператора **Dim**(**Dim** означает ***dimension***– размер)*,* а ***тип переменной*** после слова **As***.* При этом оператор объяв­ления выделяет для переменной место в памяти, по которо­му компилятор VB узнает тип данных, связанный с этой переменной.

Оператор **Dim** записывается по следующим правилам:

|  |
| --- |
| **Dim** *имена\_переменных* **As** *тип*, *имена\_переменных* **As** *тип,****…*** |

Например, следующие операторы резервируют «место» в памяти ПК для различных типов переменных:

|  |
| --- |
| **Dim LName As String**  **Dim Кол As Integer, b As Single**  **Dim Ном, d As Integer** |

Таким образом, для пере­менной с именем **LName** резервируется оперативная память*, которая* будет хранить ***стро­ковое значение***; для переменных с именем **Кол** и **b** резервируется оперативная память**,** которая будет хранить, соответственно, целые и ***вещественные значения***; для переменных с именем **Ном** и **d** резервируется оперативная память, которая хранит ***целые значения***.

Кроме оператора **Dim**, переменные можно объявить и с помощью операторов **Private** и **Public:**

|  |
| --- |
| **Private** *имена\_переменных* **As** *тип****,*** *имена\_переменных* **As** *тип…*  **Public** *имена\_переменных* **As** *тип****,*** *имена\_переменных* **As** *тип…* |

Выбор ключевого слова **Dim, Private** или **Public** определяется необходимой ***областью видимости переменной***.

Более подробно области видимости переменных и, соответственно, использование операторов **Private** и **Public** будут рассмотрены далее.

После создания переменной ей можно присваивать конкретные значения, используя оператор присваивания (символ равенства =).

***Оператор присваивания***– это фундаментальный оператор любого языка программирования. Он ***заменяет текущее значение переменной, стоящей слева от знака равенства, новым значением, которое определяется значением выражения, стоящим справа от знака равенства***. Оператор присваивания позволяет также изменять значения свойств формы и размещенных на ней управляющих элементов. Синтаксис оператора имеет следующий вид:

*Идентификатор* = *выражение.*

|  |
| --- |
| **Res = 20.56**  **X1 = А + 5 / sin(x^ 2)** |

Следующий оператор программы присваивает переменной **LName** значение «**Иванов**»:

|  |
| --- |
| **LName = "Иванов"** |

После того, как переменной **LName** присвоено значение, она может использоваться в программном коде вместо имени «**Иванов**».

Например, оператор присваивания

|  |
| --- |
| **Label1.Text = LName** |

отобразит «**Иванов**» в первой надписи (**Label1)**на форме.

Переменные могут хранить одно и то же значение на протяжении всей программы, а могут несколько раз изменять свое значение, в соответствии с программным кодом.

Следующий пример демонстрирует, как перемен­ная с именем **LName** может содержать различные текстовые значе­ния и как значение этой переменной может присваиваться свойствам объектов.

**Пример 4.2.2-1. Создать проект, в котором изменяются значения переменных.**

1. Запустить VS.
2. Выполнить команды, позволяющие создавать новый проект с именем ***Пример 2.2-1***.

В окне Конструктор появится пустая форма ***Пример 2.2-1***.

1. На ней создать интерфейс проекта ***Пример 2.2-1*** (рис.4.2.2-1).

Эта форма содержит две надписи и две кнопки. Для отображения ин­формации в каждой из этих надписей необходимо использовать пере­менные. Объекты надписей выглядят как поля потому, что свойства **BorderStyle** установлены в значение **Fixed3D**.

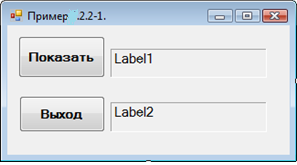


Рис. 4.2.2-1. Исходная форма проекта ***Пример 2.2-1***

1. Дважды щелкнуть мышью на кнопке формы ***Показать***.

В окне **Редактор кода** появится процедура события **Button1\_Click(…)**.

1. Чтобы объявить и использовать переменную **Name**, необходимо ввести следующие операторы:

|  |
| --- |
| **Dim Name As String**  **Name = "Петров"**  **Label1.Text = Name**  **Name ="проф. Сидоров"**  **Label4.2.Text = Name** |

Эти операторы разбиты на три группы. Первый опера­тор объявляет переменную Nameтипа **String**, используя для этого, оператор **Dim**. После ввода этой строки VS выведет под переменной Name волнистую линию, указывающую, что она была объявлена, но не используется в программе. Здесь нет ошибки – VS просто напоминает, что была создана новая переменная, кото­рую следует использовать.

Если написание программы закончено, а имя переменной останется подчеркнутым, то, возможно, где-то в коде програм­мы неправильно написано ее имя или эта переменная действительно не использовалась.

Вторая и третья строки присваивают переменной **Name** имя «**Петров**», а затем отображают это имя в первой надписи формы. Этот пример демонстрирует одно из наиболее частых применений пере­менных в программе – передачу информации в свойство. На экране все строковые значения, присваиваемые переменным, отображаются красным цветом.

Четвертая строка присваивает переменной Nameзначение   
«**проф. Сидоров**» (другими словами, она изменяет содержимое этой пе­ременной). Обратите внимание, что вторая строка длиннее, чем пер­вая, и содержит пробелы. Когда переменным присваива­ются текстовые строки или, когда строки используются в других местах, текст этих строк требуется заключать в двойные кавычки.

Наконец, необходимо помнить еще об одной важной характеристике переменных, объявленных в этой процедуре события: они поддерживают область видимости или хранят свои значения только внутри той процедуры события, в которой они используются. Далее мы узнаем о том, как объявлять переменные так, чтобы их можно было использовать в любой из процедур событий формы или других процедурах.

1. Чтобы отобразить форму, щелкнуть на вкладке **Form1.vb[Конструктор**].
2. Дважды щелкнуть мышью на кнопке **Выход**.

В окне Редактор кода появится процедура события **Button2\_Click(…)**.

1. Чтобы обозначить точку конца программы, где она остановит свое выполнение, ввести оператор **End**.

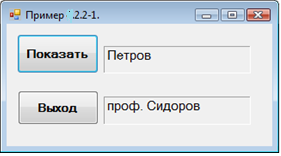
Окно редактора кода будет таким, как показано на рис. 4.2.2-2.

|  |
| --- |
| **Public Class Form1**  **Private Sub Button1\_Click(ByVal sender As System.Object,\_**  **ByVal e As System.EventArgs) Handles Button1.Click**  **Dim LName As String**  **LName = "Петров" : Label1.Text = LName**  **LName = "проф. Сидоров" :Label4.2.Text = LName**  **End Sub**  **Private Sub Button2\_Click(ByVal sender As System.Object,\_**  **ByVal e As System.EventArgs) Handles Button4.2.Click**  **End**  **End Sub**  **End Class** |

*Рис. 4.2.2-2. Программный код проекта* **Пример 2.2-1**

1. Чтобы сохранить изменения в проекте, необходимо щелкнуть на кнопке ***Сохранить все* Главной панели инструментов**.
2. Далее, чтобы запустить программу – необходимо щелкнуть на кнопке ***Начать отладку***, расположенной на Главной панели инструментов. Программа начнет выполняться, и на экране отобразится первоначальное состояние формы.
3. Щелкнуть на кнопке приложения **Показать**.

Программа продолжит выполнение: объявляется переменная, которой поочередно присваиваются два значения, и каждое из значений копируется в соответствующую надпись на форме. Результат работы программы отображается на экранной форме, показанной на   
рис. 4.2.2-3.



*Рис. 4.2.2-3. Форма проекта* **Пример 4.2.2-1***с результатом   
работы проекта*

1. Чтобы завершить выполнение программы, щелкнуть на кнопке

Выход.

**Пример 4.2.2-2.Создать проект, показывающий использование фундаментальных (базовых) типов данных.**

1. Запустить VS.
2. Выполнить команды, позволяющие создать ***проект Пример-2.2-2.***

В окне **Конструктор форм** появится пустая форма ***Пример2.2-2.***

На ней необходимо создать форму проекта ***Пример 2.2-2***(рис.4.2.2-4).

Проект Фундаментальные типы данных – это учебный проект, который будет демонстрировать работу основных типов данных. Необходимо обратить внимание, на то, где должны рас­полагаться объявления переменных так, чтобы они были доступны для всех процедур событий проекта.

Кроме того, в этом проекте используется элемент управления **ListBox*.***

ListBox – это список, в который можно записывать и из которого можно выбирать одно из предложенных значений. Значения в списке могут размещаться в одну или несколько колонок в зависимости от значения свойства MultiColumn. Если элементы списка расположены в нескольких колонках, с помощью свойства ColumnWidth можно изменить заданную по умолчанию ширину колонок.

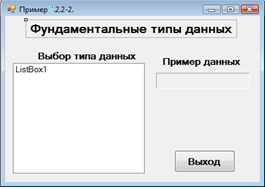


Рис. 4.2.2-4. Форма проекта **Пример 4.2.2-2**

Элементы в списке могут добавляться во время разработки проекта и программой – во время выполнения программы.

При формировании списка во время проектирования с помощью свойства Itemsвручную задается весь необходимый список. После задания всех элементов нужно нажать кнопку OK, окно редактора элементов управления закроется, а элемент управления **ListBox**будет отображать указанные значения.

1. В окне **Редактор кода** дописать программный код (рис. 4.2.2-5).

|  |
| --- |
| **Public Class Form1**  Dim Birds AsShort  Dim Insects AsInteger  Dim WorldPop AsLong  Dim Price As Single  Dim Pi As Double  Dim Debt As Decimal  Dim RetKey AsByte  **Dim UnicodeChar As Char**  Dim Dog As String  Dim Flag As Boolean  Dim Birthday As Date  **Private Sub** ListBox1\_SelectedIndexChanged(**ByVal** sender **As** \_  System.Object, **ByVal** e **As** System.EventArgs) \_  **Handles** ListBox1.SelectedIndexChanged  **Select Case** ListBox1.SelectedIndex  **Case 0**  **Birds = 12500**  **Label3.Text = Birds**  **Case 1**  **Insects = 37500000**  **Label3.Text = Insects**  **Case 2**  **WorldPop = 4800000004**  **Label3.Text = WorldPop**  **Case 3**  **Price = 899.99**  **Label3.Text = Price**  **Case 4**  **Pi = 3.1415926535**  **Label3.Text = Pi**  **Case 5**  **Debt = 7600300.5**  **Label3.Text = Debt**  **Case 6**  **RetKey = 13**  **Label3.Text = RetKey**  **Case 7**  **UnicodeChar = "Д"**  **Label3.Text = UnicodeChar**  **Case 8**  **Dog = "pointer"**  **Label3.Text = Dog**  **Case 9**  **Flag = True**  **Label3.Text = Flag**  **Case 10**  **Birthday = #3/1/1963#**  **Label3.Text = Birthday**  **End Select**  **End Sub**  Private Sub Form1\_Load(ByVal sender As System.Object, \_  ByVal e As System.EventArgs) HandlesMyBase.Load  ListBox1.Items.Add("Short") : ListBox1.Items.Add("Integer")  ListBox1.Items.Add("Long") : ListBox1.Items.Add("Single")  ListBox1.Items.Add("Decimal") : ListBox1.Items.Add("Byte")  ListBox1.Items.Add("Char") : ListBox1.Items.Add("String")  ListBox1.Items.Add("Boolean") : ListBox1.Items.Add("Date")  End Sub  Private Sub Button1\_Click(ByVal sender As System.Object, \_  ByVal e As System.EventArgs) \_  Handles Button1.Click  End  End Sub  **End Class** |

*Рис. 4.2.2-5. Программный код проекта* **Пример 2.2-2**

Необходимо отметить, что все переменные, объявленные в модуле формы, будут доступны, т.е. будут иметь область видимости во всех процедурах событий формы. Таким образом, можно установить значение переменной в одной процедуре события, а прочитать его – в другой. Обычно переменные доступны только в той процедуре собы­тия, в которой они объявлены. Чтобы сделать их доступными для всей формы, их требуется объявить в верхней части кода формы, перед событийными процедурами.

Необходимо также отметить, что в процедуре событий

**Private Sub ListBox1\_SelectedIndexChanged(…)**  используется оператор **Select Case**, который подробно будет рассмотренв **Теме 4.4**.

1. Щелкнуть по кнопке ***Начать отладку*** на **стандартной панели инструментов**.

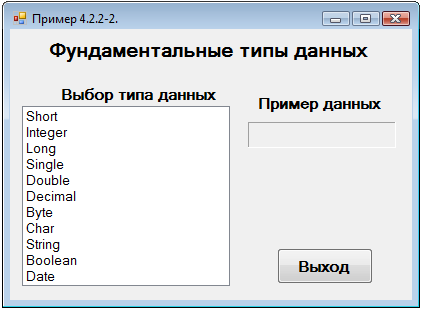


Рис. 4.2.2-6. Форма проекта **Пример.2.2-2**после загрузки формы

Появится окно приложения, изображенное на рис. 4.2.2-6.

Программа **Фундаментальные типы данных** позволяет проводить эксперименты с 11 типами данными. При щелчке на имени типа в поле спис­ка программа отображает пример данных этого типа.

1. Щелкнуть в поле списка на типе **Integer.**

В поле формы ***Пример данных*** появится число 37 500 000. Заметьте, что ни один из типов, данных **Short, Integer** и **Long** не позволяет вводить или отображать в их значениях десятичные точки.

1. Щелкнуть в поле списка на типе **Date**.

В поле формы ***Пример данных*** появится дата 3/1/1963.

1. Чтобы увидеть, как VB отображает типы данных в поле ***Пример данных***, необходимо щелкнуть в поле списка на каждом из них.
2. Чтобы остановить программу, щелкните на кнопке формы ***Выход***
3. Теперь необходимо посмотреть, как в форме объявляются базовые типы данных и как они используются в процедуре **ListBox1\_Selected\_IndexChanged(…).**
4. Рассмотрим процедуру события **Form1\_Load(…)** (рис.4.2.2-5).

Операторы этой процедуры добавляют значения в элемент управления **ListBox** при загрузке формы в начале отладки.

1. Процедуру **ListBox1\_SelectedIndexChanged(…)** обрабатывает вы­бор, сделанный в поле списка. Основой этой процедуры события является условная конструкция **Select Case**. Обратите внимание, как каждый выбор блока **Select Case** присваивает заданное значение од­ной из переменных базового типа данных, а затем присваивает зна­чение этой переменной свойству Text объекта **Label3** формы.

Попробуйте изменить данные в нескольких операторах присвоения пе­ременных и запустить эту программу, чтобы увидеть, как изменились данные. В частности, вы можете попробовать присвоить переменным значения, которые находятся вне области допустимых значений, пока­занных в таблице типов данных. Если вы сделаете такую ошибку, VB в окне Редактор кода подчеркнет некорректное значение волнистой ли­нией, и программа не запустится до тех пор, пока вы не исправите это значение. Чтобы узнать об ошибке поподробнее, поместите мышь над подчеркнутым волнистой линией значением и прочтите всплывающую подсказку, содержащую сообщение об этой ошибке.

1. Если были сделаны какие-то изменения, которые необходимо записать на диск, то щелкнуть на кнопке ***Сохранить*** на **Стандартной панели инстру­ментов**.

***Константы***. Рассмотрим правила ***объявления констант***. Если переменная в программе содержит значение, которое никогда не меняется (такое, например, как число **π***,* являющееся фиксированной математичес­кой величиной), то целесообразно хранение такого значения не в переменной, а в виде константы.

***Имя константы***– это осмысленное имя, которое заменяет неизменяемое при выполнении программы число или тестовую строку. Константы увеличивают читаемость кода программы, снижают количество программных ошибок и облегчают внесение в код глобальных изменений. Константы действуют как переменные, но при работе программы их значение не изменяется. Они объявляются с помо­щью ключевого слова **Const***,* в соответствии со следующим правилом:

|  |
| --- |
| **Const** *имя\_константы* **AS** *тип* ***=****значение\_константы* |

Например,

|  |
| --- |
| **Const Pi As Double = 3.14159265**  **Label1.Text = CStr(Pi)** |

Этот оператор создает константу с именем **Pi***,* которая может быть ис­пользована в коде программы вместо значения **3.14159265**. Чтобы сделать констан­ту доступной для всех объектов и процедур формы, по­местите этот оператор в верхней части кода формы вместе с другими объявлениями переменных и структур, область видимости которых рас­пространяется на все процедуры формы.

***Инициализаторы***– это значения, которые присваиваются переменным во время ее объявления. Это позволяет упростить код и облегчить его чтение.

|  |
| --- |
| **Dim Wd As Single = 100**  **Dim Fn As String = "Петров"**  **Dim Eps As Double= 0.0001** |

Инициализировать переменную можно только тогда, когда это единственная переменная, объявляемая в выражении **Dim**, **Public** или **Private**:

|  |
| --- |
| **Dimx, y, z As Double= 1** *' оператор вызовет ошибку* |

### 4.2.3. Выражения, встроенные функции и методы

***Выражение*** – это единица программного кода, которая определяет способ вычисления некоторого значения. Выражения в языке программирования формируются в соответствии с определенными правилами ***из констант и перемен­ных, компонент массивов, полей записей, функций и знаков операций***. Запись в тексте программного кода отличается от обычной математической записи. Операции подразделяются на следующие типы:

* арифметические;
* логические;
* строковые.

Здесь подробно рассмотрим только ***арифметические операции***, строковые операции кратко, а логические операции будут рассмотрены в **Теме 4.4.**

Таблица 4.2.3-1

|  |  |
| --- | --- |
| **перация** | **Название** |
| **+** | Сложение |
| **-** | Вычитание |
| **\*** | Умножение |
| **/** | Деление |
| **\** | Целочисленное деление (без остатка) |
| **Mod** | Остаток от деления |
| **^** | Возведение в степень |

Таблица 4.2.3-2

|  |  |  |
| --- | --- | --- |
| **Операция** | **Приоритет** | **Тип действий** |
| **()** | 1 | Вычисление в круглых скобках |
| Функция | 2 | Вычисление значения функции |
| **^** | 3 | Возведение в степень |
| Унарный **+ или –** | 4 | Унарные операции |
| **\*; /** | 5 | Операции умножения и деления |
| **\** | 6 | Целочисленное деление |
| **Mod** | 7 | Остаток от деления |
| **+; -** | 8 | Сложение и вычитание |

Принято различать ***унарные*** и ***бинарные*** операции. Унарный знак плюс «+» перед величи­ной не изменяет ее значения. Унарный знак минус «-» перед величиной изменяет ее знак на противоположный. К бинарным арифметическим операциям относятся сложение, вычитание, умножение, деление, целочисленное деление (без остатка), остаток от деления, возведение в степень.

Если в выражении используются несколько данных (переменных, констант и функций), то их ***типы должны быть идентичными либо совместимыми***. Далее об этом поговорим подробнее. Например, арифметические операции, применимые только к величинам целых и вещественных типов, приведены в табл.4.2.3-1.

В выражениях используются круглые скобки для заключения в них тех частей выражения, которые необходимо вычислить в первую очередь. Количество открывающих скобок должно быть равно количеству закрывающих. Вычисление выражений выполняется в определенном порядке, установленном круглыми скоб­ками, и в соответствии с приоритетом операций (табл. 4.2.3-2).

Операции с равным приоритетом обычно выполняют слева направо. Операция **Mod**возвращает остаток, полученный путем деления двух операндов целого типа.

**Пример 4.2.3-1. Создать проект, демонстрирующий использование операций +, -, \* и /.**

1. Запуститt VS.
2. Выполнить команды, позволяющие создавать новый проект с именем ***Пример 2.3-1. Базовые математические операции.***

В окне **Конструктор форм** появится пустая форма***.***

1. Создать форму проекта **Пример 2.3-1**(рис.4.2.3-1).

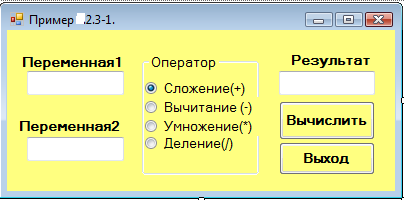


Рис.4.2.3-1. Исходная форма проекта Пример 4.2.3-1

1. Проект **Пример 2.3-1**демон­стрирует, как с вводимыми числами работают операторы сложения, вычитания, умножения и деления. Он также демонстрирует, как ис­пользовать для обработки в программе ввода пользователя объекты текстового поля, переключателя и кнопки.
2. В форме проекта **Пример 2.3-1**используются элементы управления **RadioButton**(переключатели). Так как обычно эти элементы располагаются в группах, то они позволяют выбрать одно из нескольких значений. Установка одного переключателя в группе (присвоение его свойству Checked значения *True*) автоматически сбрасывает другие переключатели, присваивая аналогичным свойствам значение *False*.
3. При размещении в форме нескольких групп переключателей, каждая логическая группа должна помещаться в отдельный контейнер, например, в элемент управления **GroupBox**. В этом случае для создания в форме группы переключателей необходимо сначала поместить в форму рамку, а затем разместить в ней поочередно необходимое количество переключателей **RadioButton**.
4. Для написания программного кода этого проекта необходимо дважды щелкнуть на кнопке **Вычислить** формы. Окно **Редактор кода** отобразит пока пустую процедуру события **Sub Button1\_Click(…)**. Ввести в нее код, приведенный на рис.4.2.3-3.
5. Дважды щелкнуть на кнопке **Выход** формы и в появившуюся процедуру события **Sub Button2\_Click(…)** ввеcти оператор **End**(рис. 4.2.3-3).
6. Щелкнуть на кнопке *Начать отладку*, на Главной панели инструментов.

Проект начнет выполняться. Эта программа отображает два текстовых поля, в которые вводятся числовые значения, группу переключателя операторов, поле, которое отображает результаты, и два объекта кнопок.

1. В текстовом поле ***Переменная1*** ввести число 100**,** а затем нажать на
2. <Tab**>**клавиатуры. Курсор ввода переместится во второе текстовое
3. поле (это поле получит фокус ввода).
4. В текстовом поле ***Переменная2*** ввести число 17.

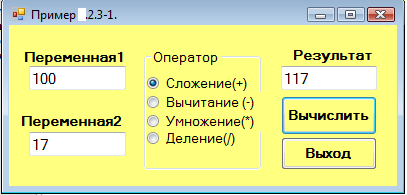


Рис. 4.2.3-2. Результат работы проекта Пример 2.3-2

1. К двум значениям применить любой из математических операторов. Например, щелкнуть на переключателе ***Сложение,*** а затем щелкнуть на кнопке ***Вычислить.***

В поле **Ре­зультат** появится число 117, как показано на рисунке 4.2.3-2.

1. Попрактикуйтесь в использовании операторов вычитания, умножения и деления для чисел, указанных в полях переменных. (Для вычисления каждой из формул необходимо щелкнуть на кнопке **Вычислить**.)

Результат появляется в поле **Результат**. Также поэксперимен­тируйте с другими числами, введя их в текстовые поля переменных. (Если хотите, можете попробовать использовать числа с десятичны­ми точками.) Для объявления соответствующих переменных исполь­зован тип данных **Double**, так что можете использовать очень боль­шие числа.

1. Далее ввести в текстовом поле **Переменная1** число 100, в текстовом поле **Переменная2**– число 0, щелкнуть на переключателе **Деление**, а затем щелкнуть на кнопке **Вычислить**.
2. Деление на ноль при математических вычислениях запрещено, так как оно дает бесконечный результат. Но VB может справить­ся с таким вычислением и отобразить в текстовом поле Результат значе­ние **Infinity** (Бесконечность). Способность обрабатывать некоторые ус­ловия деления на ноль является автоматически предоставляемой функцией VB.
3. Когда закончится процесс изучения щелкнуть на кнопке Выход.

Программа остановится.

1. Теперь, чтобы узнать, как были получены такие результаты, необходимо проанализировать код программы (рис.4.2.3-3). Код использует несколько стандартных элементов управления ввода и содержит процедуру события, которая использует переменные и операторы для вычисления простых математических формул. Программный код объявляет переменные в верхней части кода формы, и, таким образом, они могут быть использованы во всех процедурах событий **Form1**.
2. В верхней части кода также записан оператор, который объяв­ляет две переменные типа **Double**:

|  |
| --- |
| *'Объявляем переменные FirstNum и SecondNum*  **Dim FirstNum, SecondNum As Double** |

Тип **Double**был использован для того, чтобы использовать тип пере­менных общего назначения, который может работать с различными числами - целыми, числами с десятичной точкой, очень большими чис­лами, маленькими числами и тому подобным. Переменные объявле­ны в одной и той же строке с помощью сокращенной записи. Обе пере­менные **FirstNum** и **SecondNum** имеют тип **Double**, и они используются для хранения значений, вводимых в первом и втором текстовых по­лях соответственно.

|  |
| --- |
| **Public Class Form1**  **Dim FirstNum, SecondNum As Double**  **Private Sub Button2\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button4.2.Click**  **End**  **End Sub**  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles Button1.Click**  **FirstNum = TextBox1.Text**  **SecondNum = TextBox4.2.Text**  **If RadioButton1.Checked = True Then**  **TextBox3.Text = CStr(FirstNum + SecondNum)**  **End If**  **If RadioButton4.2.Checked = True Then**  **TextBox3.Text = CStr(FirstNum - SecondNum)**  **End If**  **If RadioButton3.Checked = True Then**  **TextBox3.Text = CStr(FirstNum \* SecondNum)**  **End If**  **If RadioButton4.Checked = True Then**  **TextBox3.Text = CStr(FirstNum / SecondNum)**  **End If**  **End Sub**  **End Class** |

*Рис. 4.2.3-3. Программный код проекта* **Пример 2.3-1**

Первые два оператора этой процедуры события передают данные, вве­денные в объектах текстовых полей, в переменные **FirstNum** и **SecondNum**.

|  |
| --- |
| *'Присваиваем значения текстовых полей переменным*  **FirstNum = TextBox1.Text**  **SecondNum = TextBox4.2.Text** |

Элемент **TextBox**осуществляет передачу с помощью свойства Text, которое принимает текст, введенный пользо­вателем, и делает его доступным для использования в программе.

После того, как значения текстовых полей присваиваются перемен­ным, процедура события определяет, какой переключатель был от­мечен, вычисляет математическую формулу и отображает результат в третьем текстовом поле. Первая проверка переключателя выглядит так:

|  |
| --- |
| *'Определяем выбранную кнопку и вычисляем*  **If RadioButton1.Checked = True Then**  **TextBox3.Text = CStr(FirstNum + SecondNum)**  **End If** |

В каждый момент времени из группы объек­тов может быть выбран только один объект переключателя. Чтобы определить, был ли выбран переключатель, проверьте значение свой­ства Checked. Если оно равно True, то переключатель был выбран. Если свойство Checkedравно *False*, то переключатель не был выбран.

Часто возникает необходимость выполнить в программе более сложные действия с числами. Например, может понадобиться округлить число, вычислить сложное математическое выражение или использовать в про­грамме случайные числа. Математические методы (функции), показанные в табл. 4.2.3-3, могут помочь в работе с числами в ваших формулах. Эти методы (процедуры) предоставляются MS .NET Framework– ***библиотекой клас­сов***, которая позволяет выполнить множество часто возникающих задач программирования, требуемых реализовать в проекте.

Известно, что платформа .NETF rameworkорганизована в виде библиотек классов, которые могут быть включены в проекты программирования по их именам с помощью опера­тора **Imports***.* Этот процесс достаточно прост, и далее мы поэкспериментируем с мате­матическими методами (процедурами) из класса **System.Math.**

В табл. 4.2.3-3 представлен неполный список математических методов класса **System.Math**. Аргумент **x**в этой таблице представляет число, переменную или оператор, которые требуется передавать в метод для обработки. При использовании каких-либо из этих методов необходимо убеди­ться, что оператор **Imports System.Math** находится в окне **Редактор кода** в самом начале программного кода.

Таблица 4.2.3-3

|  |  |
| --- | --- |
| **Функция**  **(метод)** | **Возвращаемое значение** |
| **Abs(x)** | Абсолютное значение x (т.е. игнорируется знак, Abs(-5)=5 ) |
| **Acos(x)** | Значение арккосинуса х |
| **Asin(x)** | Значение арксинуса х |
| **Atan(x)** | Значение арктангенса x. |
| **Ceiling(x)** | Целое число, которое больше или равно аргументу |
| **Cos(x)** | Значение косинуса x |
| **DivRem(x,y,r)** | Значение остатка (r) от целочисленного деле­ния |
| **E** | е – oснование натурального логарифма – 4.2.71828182845905 |
| **Exp(x)** | Экспонента - ех |
| **Fix(x)** | Целую часть числа (дробная часть отбрасывается) |
| **Floor(x)** | Целое число, которое меньше или равно аргументу |
| **Int(x)** | Значение целой части **х** (ближайшее меньшее) |
| **Log(x[,n])** | Логарифм по основанию n |
| **Log10(x)** | Десятичный логарифм |
| **Max(x, y)** | Максимальное из двух чисел |
| **Min(x, y)** | Минимальное из двух чисел |
| **PI** | π – 3.1415265358979 |
| **Pow(x, y)** | Возведение x в степень y |
| **Rnd( )** | Равномерное случайное число на отрезке [0;1] |
| **Round(x, n)** | Округление до n знаков после запятой |
| **Sign(x)** | Знак числа х |
| **Sin(x)** | Значение синуса x |
| **Sqrt(x)** | Квадратный корень из х |
| **Tan(x)** | Значение тангенса x |
| **Truncate(x)** | отбрасывает дробную часть |

Прокомментируем некоторые методы.

Функция **Log(x[, n])** поддерживает один аргумент (натуральный логарифм) или два аргумента (логарифм по основанию **n**):

|  |
| --- |
| **Dim result As Double = Math.Log(10)** *' = 4.2.30258509299405* **result = Math.Log(1000,10) *' = 3***  **result = Math.Log10(1000)** *'=3* |

Методы **Min(x,y)**и **Max(x,y)**вычисляют минимальное и максимальное значения, соответ­ственно, и используются для работы со всеми числовыми типами данных.

Метод **Floor(x)** возвращает целое число, которое меньше или равно аргументу мето­да, в то время как метод **Ceiling(x)** возвращает целое число, которое больше или равно аргументу метода. Метод **Truncate(x)**просто отбрасывает дробную часть и, следовательно, работает отлично от метода **Ceiling(x)**, если аргумент отрицателен:

|  |
| --- |
| **result = Math.Ceiling(4.2.5)** *' = 3*  **result = Math.Floor(-1.5)** *' = -2*  **result = Math.Truncate(-1.5)***' = -1* |

Метод **DivRem(x,y,r)**выполняет целочисленное деле­ние и возвращает остаток (**r**) в третий аргумент:

|  |
| --- |
| **Dim remainder As Long**  **Dim result As Long= Math.DivRem(20, 3, r)** *' = 2* |

Напомним, если логарифм имеет не натуральное основание, то его можно преобразовать к натуральному логарифму – **Logab=Log(b)/Log(a).**

Значение аргумента **х** в тригонометрических функциях дается в радианах.

Обычно по традиции, используемые в языках программирования библиотечные функции, называют ***«встроенными функциями»***. На самом деле они представляют собой стандартные методы (процедуры) из библиотеки VB или из библиотеки классов Microsoft .NET Framework, которым передается один или несколько аргументов и которые возвращают вычисленное значение через имя функции.

Обращение ко всем функциям всегда происходит по их именам, за которыми в скобках следуют аргументы. Вычисле­ние функций в арифметическом выражении предшествует выполне­нию арифметических операций. Аргументом встроенной функции может быть любое ариф­метическое выражение. Кроме того, если в начале программного кода отсутствует оператор **Imports System.Math**, то перед обращением к функциям необходимо указать имя класса с точкой, например, **Math.Sin(x).**

***Обратите внимание, что в табл.4.2.3-3 описаны две системные константы* (E** и **PI)*.***

**Пример 4.2.3-2. Создать проект, использующий метод класса   
System.Math для вычисления квадратного корня.**

1. В меню Файл выполнить команду ***Создать проект***.

Появится диалоговое окно **Создать проект**.

1. Создать новый проект типа с именем **Пример 4.2.3-2** (Встроенные математические функции).

В окне Конструктор появится пустая форма.

1. Щелкнуть на элементе управления **Button**окна **Панель элементов** и создать в верхней части формы объект кнопку.
2. Щелкнуть в окне **Панель элементов** на элементе управления **TextBox** и отобразить под объектом кнопка два объекта текстового поля.
3. Установить свойство Textобъекта кнопки равным **Квад­ратный корень**.
4. Вызвать окно **Редактор кода**.
5. В самом верху окна **Редактор кода**, над оператором **PublicClassForm1**, ввести оператор программы: **Imports System.Math**

Оператор **Imports** добавляет в проект библиотеку объектов, свойств и методов. Этот оператор должен быть первым оператором программы - он должен идти даже перед переменными, которые объявляются для всей формы, и перед оператором **Public Class Form1**, который в VB создается автоматически. Конкретной библиотекой классов, которую мы выбрали, является **System.Math**, которая является коллекцией объек­тов, свойств и методов, предоставляемой .NET Frameworkдля выпол­нения математических операций.

1. Спуститься в окне **Редактор кода** вниз и в процедуру события   
   **Button1\_Click(…)**между операторами **Private Sub** и **End Sub**добавить следующий код:

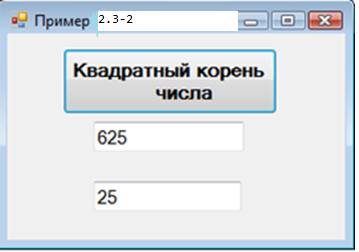
|  |
| --- |
| **Dim rg As Double**  **Dim Result As Double**  **Arg = TextBox2.Text**  **Result = Sqrt(Arg)**  **TextBox1.Text = Result** |

Эти операторы объявляют переменную типа **Double**с именем **Result**, используют метод **Sqrt()**для вычисления квадратного корня из числа, введенного в поле **TextBox2.Text**, и присваивают значение переменной **Result**свойству Textобъекта текстового поля **TextBox1**так, что в нем отображается результат вы­числения.

1. Чтобы сохранить изменения, щелкнуть на кнопке ***Сохранить Все*** стандартной панели инструментов. В качестве места сохранения укажите необходимую папку (каталог)***.***
2. Щелкнуть на кнопке ***Начать отладку*** на стандартной панели инструментов.
3. Введите число 625 в поле **TextBox2.Text**.
4. Щелкнуть на кнопке формы **Квадратный корень**.

VB вычислит квадратный корень из 625 и отобразит в тек­стовом поле

результат этого вычисления (рис. 4.2.3-8).



*Рис. 4.2.3-4. Форма проекта* **Пример 2.3-2**

Рассмотрим члены базовых числовых классов, а именно, методы и поля (свойства), которые будут полезны при разработке проектов лабораторных работ.

Для всех числовых типов VBи всех соответствующих классов   
(табл. 4.2.2-1) определены статические поля **MinValue** и **MaxValue**. Эти поля возвращают наименьшее и наибольшее значение, которое можно присвоить переменной соответствующего типа:

|  |
| --- |
| *'Наибольшее значение, которое можно хранить*  *'в переменной типа Double.*  **TextBox1.Text= Double.MaxValue***' = 1.79769313486232Е+308* |

Числовые классы, которые поддерживают значения с плавающей точкой, а имен­но, классы **Single** и **Double**, имеют несколько дополнительных статических свойств, предназначенных только для чтения.

Свойство Epsilon возвращает наименьшее по­ложительное (не равное нулю) число, которое можно хранить в переменной данно­го типа:

|  |
| --- |
| **TextBox1.Text= Single.Epsilon***'= 1.401298E-45*  **TextBox4.2.Text= Double.Epsilon***'= 4.94065645841247E-324* |

В классах **Single**и **Double**, к тому же, определены статические методы, которые позволяют проверять то, что переменные их типа содержат специальные значения, такие как **IsNegativelnfinity(), IsPositivelnfinity()** и **IsNaN( )**.

В классе **Microsoft.VisualBasic.Interaction** определено множество полезных команд и методов взаимодействия программ, включая **Beep(),   
InputBox(), MsgBox()**и другие. Эти методы определены глобально, поэтому при их использовании можно не указывать имя класса.

Кроме числовых данных, при написании программного кода активно используются строковые и символьные данные. Подробно они будут рассмотрены в **Теме 4.9**, где будут описаны свойства и методы классов **System.String**и **System.StringBuilder.NET**. Здесь кратко рассмотрим строковые типы данных, так как они будут использоваться почти во всех программных кодах проектов для ввода и отображения данных на форме.

Основная операция со строка – это операция конкатенации – & (+):

|  |
| --- |
| **Dim s As String = "ABC"**  **s = s&"D"** *'Результат строка - ABCD* |

Объявленная строковая переменная, которой еще не присвоено никакое значение, содержит значение Nothing или, точнее, не указывает на какую-либо последовательность символов в памяти. Такая переменная отличается от переменной, которая равна пустой строке:

|  |
| --- |
| **Dim s As String =**"" *'Длина строки равна 0*  **Dim t As String***'Длина строки не определена, 'а значение = Nothing* |

Необходимо обратить внимание, что VB не предлагает способа ввода в строку непечатных символов, например, символов табуляции или новой строки. Строковые константы, такие как **vbCrLf, vbTab**и другие, поддерживаются как свойствами класса **Microsoft.VisualBasic.Constants.** Этот класс обозначен как глобальный, поэтому в программный код необязательно включать имя этого класса.

В качестве альтернативы можно использовать свойства класса   
**Microsoft.VisualBasic.ControlChars.** Поскольку этот класс не объявлен как глобальный, имя класса нужно указывать в программном коде (если только не использовалось выражение **Imports** для импорта всего этого класса). Имена констант в этом классе такие же, как и в классе   
**Microsoft.VisualBasic.Constants**за исключением того, что в них не используется префикс **vb**.

Класс **ControlChars** содержит константы **Back, Cr, CrLf, FormFeed, NewLine, NullChar, Quote, Tab**и**VerticalTab**. Наиболее используемые константы представлены в табл.4.2.3-4.

Таблица 4.2.3-4

|  |  |  |  |
| --- | --- | --- | --- |
| **Константы**  **Класса Constants** | **Константы**  **Класса  ControlChars** | **Код символа** | **Описание** |
| **vbCrLF** | **CrLF** | Chr(13)+Chr(10) | Символы возврата каретки и  перевода строки |
| **vbCr** | **Cr** | Chr(13) | Символ возврата каретки |
| **vbLf** | **Lf** | Chr(10) | Символ перевода строки |
| **vbNewLine** | **NewLine** | Chr(13)+Chr(10) | Символ новой строки |

Чтобы включить эти символы в строку, необходимо использовать конкатенацию строк. Например,

|  |
| --- |
| **Dim s As String = "МТУСИ"&ControlChars.NewLine**  **s = s&"Кафедра" &"ВМ и П" &vbCrLF** |

Класс **Microsoft.VisualBasic.Strings**содержит большинство строковых функций язы­ка VB, включая **Asc, Chr, ChrW, Filter, Format, FormatCurrency, FormatDateTime, FormatNumber, FormatPercent, InStr, InStrRev, Join, LCase, Left, Len, LTrim, Mid, Replace, Right, RTrim, Space, Split, StrComp, StrReverse, Trimи UCase**. Эти методы определены глобально, поэтому для их вызова необязательно указывать полное имя класса, хотя нужно указывать полное имя связанных с ними констант. Подробнее эти функции будут рассмотрены в **Теме 4.9**.

### 4.2.4. Преобразование данных в выражениях и операторе присваивания

У многих программистов при работе на VBчасто возникают проблемы с правильным преобразованием вещественных чисел или же строк, содержащих числа, к целым числам.

Рассмотрим более подробно проблему ***преобразования данных в выражениях и операторах присваивания.***

В процессе рассмотрения примеров мы уже встречались с проблемой преобразования данных. При работе с данными в VB нужно иметь в виду поддерживаемые в VS преобразования.

Во-первых, это ***расширяющие, не приводящие к потерям данных, преобразования***, например,

**Integer → Long, Decimal, Single, Double.**

Во-вторых, это ***сужающие – преобразования (с возможностью потерей данных) в тип данных, содержащий меньшие значения данных***, например,

**Single → Integer;**

**Double → Integer*.***

Потери данных, связанных с сужающими преобразованиями, можно из бежать, используя установки

**Option Strict On**

**Option Explicit On.**

Напомним, что первая установка (**Option Strict On**) разрешает все расширяющие преобразования, а все сужающие преобразования заставляет программиста делать осознанно, т.е. явно вызывая нижеописанные функции преобразования типов. Вторая установка (**Option Explicit On**) принуждает программиста объявлять все переменные до их использования в программе.

Обратите внимание, что в лабораторных работах приведенные выше установки обязательно необходимо использовать.

Приведем список функций языка VB, являющихся ключевыми словами, и, которые позволяют осуществлять явные преобразования типов:

* **CBool(x)** - возвращает логическое значение. В качестве параметра **x**функции может быть передано условие или числовое значение.
* **CByte(x)**- возвращает значение типа **Byte**. В качестве параметра **x**передается соответствующая строка или числовое значение.
* **CChar(x)** - возвращает символ. В качестве параметра x передается целое число, находящееся в промежутке от 0 до 65 535.
* **CDate(x)** - возвращает значение типа **Date**. В качестве параметра **x** можно использовать любое принятое обозначение даты.
* **CDbl(x)** - преобразовывает значение параметра **x** к типу **Double**.
* **CDec(x)**- применяется для преобразования параметра **x** к типу **Decimal**.
* **CInt(x)**- преобразовывает значения параметра **x** к целочисленному типу **Integer**. Если в качестве параметра **x**передано дробное число, то дробная часть просто округляется по правилам математики.
* **CLng(x)** - возвращает значение типа **Long**. Обработка параметра **x** производится по образцу функции **CInt(x)**.
* **CShort(x)** - преобразовывает значение параметра **x** к типу **Short**.
* **CSng(x)** - применяется для преобразования значения параметра **x** к типу **Single**.
* **CStr(x**) - используется для преобразования данных в строковый тип **String**.

Кроме «внутренних» функций языка VB для явных преобразований можно использовать классы и модули как VB, так и .NET.

В модуле **Microsoft.VisualBasic.Conversion** определено множество методов преобразования значений, включая:

* **Int()** и **Fix()** – функции возвращают целую часть числа. Предполагается, что аргумент имеет тип **Double** или является числовым выражением. Возвращается тип **Double**. А уже этот **Double** затем можно преобразовать в **Integer** или **Long**. Главное, чтобы число уместилось в диапазон нужного вам типа. Разность при использовании **Int()**и **Fix()**заметна только лишь при отрицательном значении аргумента. Дело в том, что **Int()** всегда возвращает ближайшее меньшее целое число, а **Fix()** просто отбрасывает дробную часть. То есть, для Int(4.5)и**Fix(4.5**) возвращает 4, тогда как для **Int(-4.5)** возвращает – 5, а для **Fix(-5)** – -4.
* **Val(x)** – преобразует значение параметра x строкового типа **String** к числовому значению типа **Double**.
* **Str(x**)– преобразует число в строку, причем перед положительным числом вставляется пробел.

|  |
| --- |
| **Dim Number As Single**  **Number = Int(99.8)** *' Возвращает -99*  **Number = Fix(99.8)** *' Возвращает -99*  **Number = Int(-99.8)***' Возвращает -100*  **Number = Fix(-99.8)** *' Возвращает -99*  **Number = Int(-99.2) *' Возвращает -100***  **Number = Fix(-99.2)** *' Возвращает -99* |

Эти функции преобразовывают числовые аргументы в целое число типа **Long** и **Integer** соответственно. Дробная часть при этом округляется по правилам математики. Таким образом, результатом преобразования **CInt(1.5)** или **CLng(1.5)** будет целое число 4.2. А если аргументом будет 1.4, то в результате получим 1. Знак числа при этом роли не играет, то есть -1.5 будет преобразовано в -2, а -1.4 – в -1.

При преобразовании из строки, содержащей число, важно помнить, что функции преобразования используют национальные настройки для получения символа разделения целой и дробной частей числа. Поэтому, если у вас разделителем целой и дробной частей числа является точка «.», а в строковом представлении числа будет фигурировать запятая, то вы получите ошибку несовпадения типов при попытке преобразовать такую строку в целое число (кстати, эта ошибка возникнет и при попытке использования **CDbl()**или **CSng()** для преобразования строки в вещественное число, а также и при использовании **Int()** или **Fix()**).

***Функции* Val(), CDbl()**и**CSng()*.*** При использовании этих функций имеются свои нюансы. **CDbl()** и **CSng()**аналогичны **CLng()** и **CInt().   
CDbl()**преобразовывает аргумент в вещественное число типа **Double**, а   
**CSng()** соответственно в **Single**. Естественно, надо обращать внимание на попадание аргумента в диапазон представления и на правильность разделителя целой и дробной частей. Для функций **CDbl()** и **CSng()**разделителем целой и дробной части числа обязательно должен быть символ запятая «,», а для функции **Val()** разделителем целой и дробной частей числа является точка «.». Функция **Val()** возвращает число, содержащееся в строке - аргументе как числовое значение типа **Double.** Данная функция будет пытаться преобразовать строку до тех пор, пока не встретит символ, который не может быть распознан как часть числа. В частности, функция прекрасно принимает точку в качестве разделителя, а вот пробелы просто игнорирует.

Таким образом, функция **Val()**преобразует строки в соответствии с табл.4.2.4-1.

Ещё раз обратите внимание, что функции **Val(x)** и **Str(x**) при преобразовании в качестве разделителя целой и дробной частей числа функции **Val(x)** распознают только точку, а функция **СDbl(x)** и ей подобные – запятую, но та и другая функция игнорирует пробелы в числе.

Рассмотрим несколько примеров:

|  |
| --- |
| **DimaDblAs Double**  **DimanIntAs Integer**  **aDbl = 2345.5678**  **anInt = CInt(aDbl)** *' ПеременнаяanIntбудетравна 2346.*  **Dim aDbl1, aDbl2 As Double**  **Dim aLng1, aLng2 As Long**  **aDbl1 = 25427.45 : aDbl2 = 25427.55**  **aLng1 = CLng(aDbl1)** *' ПеременнаяaLng1 будетравна 25427.*  **aLng2 = CLng(aDbl2)** *' Переменная aLng2 будет равна 25428.*  **Dim aDouble1, aDouble2 As Double**  **Dim aSingle1, aSingle2 As Single**  **aDouble1 = 75.3421105 : aDouble2 = 75.3421567**  **aSingle1 = CSng(aDouble1)**  *'Переменная aSingle1 будетравна 5.34211.*  **aSingle2 = CSng(aDouble2)**  *' Переменная aSingle2 будет равна 75.34216.*  **DimaDoubleAs Double**  **Dim aStringAs String**  **aDouble = 437.324**  **aString = CStr(aDouble)**  *'ПеременнаяaStringбудетравна 437.324.*  **aDouble = CDbl(**"**437,324**")  *'Переменная aDouble будет равна 437,324.*  **aDouble = Val(**"**437.324**")  *'Переменная aDouble будет равна 437.324.* |

Таблица 4.2.4-1

|  |  |  |
| --- | --- | --- |
| **Строка** | **Число** | **Примечания** |
| "     1 2 34569" | 1234569 | После отбрасывания пробелов получаем целое число 1234569. |
| "     1 2 345,69" | 12345 | После отбрасывания пробелов функция пытается разобрать число и останавливается на символе «,», и возвращает 12345. |
| "     1 2 345.69" | 12345.69 | В отличие от предыдущего случая, здесь символ «.» является законным разделителем целой и дробной частей числа, поэтому результатом будет 12345.69. |
| "    12 34569number" | 1234569 | Здесь разбор числа будет остановлен на первой встреченной букве «n», а все, что до нее, преобразовывается в число 1234569. |

***Округление.*** Для округления используется функция **Round()**. Она возвращает число с заданным количеством цифр после запятой. Однако есть один нюанс. Функция **Round()** именно округляет (что собственно видно из названия). То есть, вызов **Round(1.15,1)** даст результат 1.4.2. Иногда возникает ситуация, когда необходимо отбросить лишние цифры, то есть для числа 1.15 получить 1.1. Тогда надо пользоваться конструкцией вида: **Fix(10\*n)/10**.

Здесь вместо 10 должна использоваться степень 10, соответствующая количеству оставляемых цифр после запятой. Универсальный вариант будет таким: **Fix(10^ЧислоЦифрПослеДесТочки\*n)/10^ЧислоЦифрПослеДесТочки.**

Обратите внимание, что здесь используется именно функция **Fix(),** а не **Int()**, которая для отрицательных чисел будет работать некорректно.

В **модуле Microsoft.VisualBasic.String**определенно множество методов работы со строками, в том числе**Format(x, строка символов)** - преобразует числовое значение параметра x в заданный формат, который определяется строкой символов. Любое числовое значение преобразовывается в символьное представление данного числа.

Таблица. 4.2.4-2

|  |  |
| --- | --- |
| **Символы** | **Пояснение** |
| **#** | Число в обычном виде и без дробной час**ти** |
| 0 | Число в обычном виде и без дробной части |
| #.### | Число в обычном виде. После запятой пользователь видит не больше 3 знаков, целая часть дроби будет показана полностью |
| 0.000 | Число в обычном виде. После запятой пользователь видит не больше 3 знаков, целая часть дроби будет показана полностью |
| E | Число показывается в экспоненциальной форме |

Возможные значения строки символов функции **Format()**и их описание приведены в табл. 4.2.4-2.

Если параметр функции**Format()** *– строка символов* опущен, то функция производит тоже действие, что и функция **С[Str](http://www.kbyte.ru/Refbooks/show.aspx?id=261)()** для численного выражения. Положительные числа, преобразуемые в строки, теряют пробел слева, зарезервированный для знака. Если положительное число преобразовано в строку функцией **С[Str](http://www.kbyte.ru/Refbooks/show.aspx?id=261)()**, то пробел слева сохраняется. Например,

|  |
| --- |
| **Format(2, "000000")** *'возвращает 000002* **Format(10, "0.00")** *'возвращает 10.00* |

Для всех числовых типов VB и всех соответствующих классов .NET определен метод **ToString(x)**, который преобразует числовое значение в строку. Этот метод особенно по­лезен, когда строковое представление числа нужно добавить к другой строке, например,

|  |
| --- |
| **Dim myValue As Double = 123.45**  **Dim res As String = "The final value is"&myVaIue.ToString()** |

Метод **ToString(x)** по умолчанию используются настройки, связанные с текущим потоком символов. Например, в качестве десятичного разделителя он использует точку, если национальные настройки – это русский язык.

|  |
| --- |
| *'Вывод PI с 5 знаков после десятичной точки (всего 6 цифр)*  **Dm d As Double = Math.PI**  **TextBox1.Text= d.ToString("G6")** *'рез-т 3.14159 - всего 6 цифр* |

Все числовые классы поддерживают перегруженную форму метода **ToString()**, которая позволяет использовать строку форматирования целых действительных чисел в однотипные строки:

|  |
| --- |
| **Dim intValue As Integer = 12345**  **TextBox1.Text = intValue.ToString("##,##0.00")***'= 12 345.00*  **Dim sngValue As Single = 12345.5**  **TextBox4.2.Text = sngValue.ToString("##,##0.00")** *'= 12 345.50* |

Запятая в строке форматирования означает отделение каждых трех цифр в числе пробелом, а точка – представление дробной части числа.

Для явного преобразования любого числового типа в строку, кроме функций **Str()** и **CStr()**, можно использовать метод **ToString()**класса **Convert**:

|  |
| --- |
| **Dim Число As Double**  **Число = 3.14**  **Dim СтрокаAs String = Convert.ToString(Число)** |

Напомним также, что обратное преобразование из строки в число можно выполнить или с помощью функций **Val(), CDbl(), CInt(),** или с помощью соответствующих методов класса **Convert – ToDouble(),  
ToInt32()** и других. Причем список методов можно увидеть с помощью технологии ***IntelliSense*** при вводе точки после **Convert**.

### 4.2.5. Пример создания и отладки проекта «Вычисление периметра и площади треугольника»

1. **Название работы:**

Вычисление периметра и площади треугольника.

1. **Задание на разработку проекта Проект 4.2.5:**

Сформулируем содержательную постановку задачи:

Создать приложение на VB с именем **Проект 4.2.5**, которое должно вычислять периметр и площадь треугольника по заданным значениям трех его сторон:

Задано: **А, В, С** – значения сторон треугольника.

Требуется определить: **Р** – периметр треугольника,

**S** – площадь треугольника.

Ограничения на значения исходных данных и их соотношения**:**

А>0, В>0, C>0, А+В>С, А+C>В, В+C>А одновременно.

**Проект4.2.**5 должен иметь простой интерфейс пользователя:

* две кнопки управления;
* три поля, отображающие входные данные;
* два поля, отображающие выходные данные.

1. **Формализация и уточнение задания:**

Для решения задачи существуют известные формулы:

**Р = А + В + С; S = (формула Герона),**

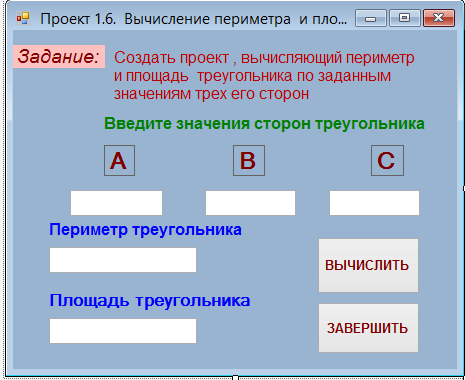


Кроме того, при разработке интерфейса надо учитывать следующее:

* строка заголовка формы должна содержать название работы   
  «Проект 4.2.5. Вычисление периметра и площади треугольника»;
* для ввода исходных данных необходимо использовать три текстовых поля;
* вычисления должны производиться при нажатии на кнопку   
  **ВЫЧИСЛИТЬ**;
* для вывода результатов следует использовать текстовые поля (два поля);
* для завершения приложения должна быть кнопка **ЗАВЕРШИТЬ**;
* приложение должно содержать поясняющие надписи;
* зоны ввода и вывода данных должны быть отображены визуально.

Принимая во внимание вышеизложенное, Форма будет соответствовать

рис.4.2.5-1.



*Рис. 4.2.5-1. Форма проекта* **Проект 4.2.5**

*Вычисление периметраи площади треугольника*

1. ***Разработка приложения (проекта):***
2. **Создание нового проекта:**

* Запустить VS;
* Создать новый проект, т.е. выполнить команду ***Создать проект*** элемента **Главного меню Файл**.

На экране появится диалоговое окно **Создание проекта**:

* Выбрать шаблон **Приложение Windows Forms** в диалоговом окне Создание проекта.
* Ввести в текстовое поле ***Имя*** нижней части диалогового окна   
  **Создание проекта** имя проекта ***Проект 4.2.6.***
* Щелкнуть на кнопке ОК диалогового окна Создание проекта.

VS присвоит проекту имя **Проект 4.2.6** и отобразит **Основное окно** с различными панелями, среди которых присутствует окно

Конструктора **форм** с пустой формой **Form1**. Эта форма будет использоваться для создания интерфейса пользователя.

1. **Создание интерфейса пользователя:**

Изменить размер формы, для этого необходимо поместить указатель мыши над нижним правым углом формы так, чтобы он изменился и принял форму указателя изменения размера, а затем перетащить этот угол так, чтобы увеличить размер формы для размещения на ней объектов проекта. При изменении размера формы в окне **Конструктора форм** могут появиться полосы прокрутки, которые позволяют получить доступ ко всей созданной форме. В зависимости от разрешения экрана и открытых инструментов VS, можно не увидеть всю форму целиком. Задать размер формы, примерно равный размеру формы, показанной на рис. 4.2.5-1.

* Чтобы увидеть всю форму без помех, если это необходимо, можно изменить размер других панелей инструментов или закрыть их.
* Добавить на форму объект **Button1**, для чего дважды щелкнуть в окне **Панель элементов** на **Button**. VS создаст на форме кнопку с размерами по умолчанию и именем **Button1**. Причем эта кнопка выделена и, вокруг имеются манипуляторы изменения размера.

Обратить внимание на название кнопки – оно встретится при написании кода программы. Когда VB находится в режиме конструктора, объекты на форме перемещаются с помощью перетаскивания их мышью, а их размеры изме­няются с помощью манипуляторов изменения размера.

* Добавить на форму второй объект **Button2**, для чего щелкнуть (один раз) в окне **Панель элементов** на элементе управления **Button**, а затем переместить указатель мыши в область формы **Form1**. Указатель мыши примет форму перекрестия с пиктограммой кнопки. Перекрестие предназначено для того, чтобы нарисовать прямоугольную фигуру кнопки на форме, и этот метод используется как альтернатива двойному щелчку, создающему элемент управления с размером по умолчанию.
* Перетащить указатель вниз и вправо. Чтобы завершить образование второй кнопки, необходимо отпустить кнопку мыши и посмотреть на привязку **Button2** к форме.

Изменить размер объекта кнопки так, чтобы она стала такого же размера, как и первая кнопка, а затем переместите ее на форме, расположив ниже первой кнопки.

В любой момент можно удалить объект и начать заново – выбрав этот объект на форме, а затем нажав на клавишу <Del> клавиатуры.

Попрактиковаться в создании и удалении объектов интерфейса пользователя. Форма должна выглядеть примерно так, как показано рис. 4.2.5-2.

* Добавить на форму текстовые надписи, для чего необходимо дважды щелкнуть мышью в окне **Панель элементов** на **Label*.***

VS создаст на форме экземпляр элемента управления **Label1.**

Перетащить элемент **Label1**влево от двух объектов кнопок.



*Рис. 4.2.5-2. Форма проекта с двумя кнопками*

* Создать второй объект надписи, дважды щелкнув мышью в окне **Панель элементов** на элементе управления **Label.**

VS создаст на форме элемент **Label4.2**.

* Создать третий объект надписи **Label3,** и снова дважды щелкнуть мышью на элементе управления **Label*,*** чтобы создать третий объект надписи **Label3.**



Рис. 4.2.5-3. Форма проекта с двумя кнопками и четырьмя надписями

* Создать четвертый объект надписи **Label4**, для чего необходимо использовать элемент управления **Label** для добавления к форме описательной надписи.
* Переместить третий и четвертый объекты надписей вправо от второго, оставив между ними небольшое расстояние.

Форма будет иметь вид примерно такой, как показано на   
рис. 4.2.5-3.

* Снова дважды щелкнуть мышью на элементе управления **Label**, чтобы создать пятый объект надписи **Label5**.
* Создать шестой объект надписи **Label6**.
* Переместить пятый и шестой объекты надписей вниз от второго, третьего и четвертого.
* Добавить на форму текстовые поля, которые позволят отображать на форме входные и выходные значения, для чего дважды щелкнуть мышью в окне Панель элементов на элементе
* управления **TextBox**. VS создаст на форме экземпляр объекта **TextBox1**.
* Далее поместить элемент **TextBox1**внизу, под элементом **Label2**.
* Таким же образом необходимо создать на форме элементы **TextBox2** и **TextBox3**, разместив их под элементами **Label3** и **Label**4.
* Создать на форме элементы **TextBox4** и **TextBox5**, разместив их под элементами **Label5** и **Label6** соответственно.
* Форма будет иметь вид примерно такой, как показано на   
  рис. 4.2.5-4.

**

*Рис. 4.2.5-4. Форма проекта с двумя кнопками и шестью надписями*

1. **Установка свойства объектов**

Первоначально все элементы управления и форма имеют значения свойств, назначенные системой (по умолчанию).

Свойства объектов можно изменить при помощи выбора объектов на форме и изменения их значений в окне **Свойства**:

* Установить свойства первой кнопки, для этого щелкнуть на кнопке **Button1**формы.

Кнопка выделится и будет окружена манипуляторами.

* Затем щелкнуть на строке заголовка окна **Свойства**.

Если окно **Свойства** скрыто, щелкнуть на команде ***Свойства***

***элемента*** главного меню **Ви**д или нажать на клавишу <F4>.

Изменить (если требуется) размер окна **Свойства** так, чтобы в

нем было достаточно места для отображения имен свойств и их

значений.

Когда привыкнете устанавливать свойства, то сможете

использовать окно Свойства без увеличения его размеров.

* В данный момент окно Свойства содержит список значений свойств для первой кнопки **Button1**. Они включают значения для цвета фона, текста, высоты шрифта и ширины кнопки и др. Так как имеется много свойств, VS организует их в категории и отображает в виде структуры. Если необходим увидеть свойства из какой-то категории, щелкнуть на знаке «плюс» (+) рядом с названием категории.
* Прокрутить список окна **Свойств**а так, чтобы увидеть свойство Text, расположенное в категории ***Внешний вид***.
* Дважды щелкнуть мышью в окне **Свойства** в левом столбце свойства Text.

В окне **Свойства** будет выделено текущее значение

свойства Text**Button**1.

Необходимо ввести текст **ВЫЧИСЛИТЬ** и нажать клавишу клавиатуры <Enter>.

В окне **Свойства** Text изменит свое значение на **ВЫЧИСЛИТЬ**, и

после выхода из окна **Свойств**а этот же текст появится на самой

кнопке.

Установить свойства второй кнопки **Button2** другим способом. Для этого раскройте список **Объек**т в верхней правой части окна **Свойств**а.

Появится список объектов интерфейса программы.

В этом списке щелкнуть на Button2 из списка всех

используемых элементов управления System.Windows.Forms.

В окне **Свойства** появятся значения свойств второй кнопки, а VS

выделит **Button**2 на форме.

Дважды щелкнуть мышью на текущем свойстве Text «Button2» и введите текст «**ЗАВЕРШИТЬ**», а затем нажать <Enter>.

Название второй кнопки формы изменится на «**ЗАВЕРШИТЬ**».

Использование списка **Объек**т является удобным способом переключения между объектами проекта. Можно переключаться между объектами формы, просто щелкая на них.

Установить свойства **Label2**. Для этого щелкнуть на **Label2**, а затем в окне свойству Text присвоить значение «А», свойству TextAlign(Выравнивание) значение 2-Center (Выравнивание по центру), свойству Font (Шрифт) значения – Arial, жирный, размер 12, свойству ForeColor(Цвет метки) значение Maroon (Коричневый), свойству BorderStyle (Стиль рамки подписи) значение FixedSingle (Видимая граница).

Аналогичным образом необходимо установить свойства для **Label3** и **Label4**.

Установить свойства **Label1**. Для этого щелкнуть на **Label**1, а затем в окне **Свойства** свойству Text присвоить значение «**Введите значения сторон треугольника**», свойству TextAlign (Выравнивание) значение 2-Center(Выравнивание по центру), свойству Font(Шрифт) значения –Arial, жирный, размер 12, свойству ForeColor(Цвет метки) значение Green(Зеленый), свойству BorderStyle(Стиль рамки подписи) значение None(Без границ).

Аналогичным образом необходимо установить свойства для **Label5** и **Label6**.

Установить свойства элементов **TextBo**x: **TextBox1 – TextBox5**. Для этих объектов примем значения свойств по умолчанию.

Изменить заголовок формы можно, щелкнуть в области **Form1**, а затем в правом столбце свойства Text окна **Свойства**, и ввести текст «**Вычисление периметра и площади треугольника**». В верхней части формы напишите текст задания (рис. 4.2.5-1).

1. **Составление таблицы используемых объектов и их свойств.**

Используемые объекты и их свойства свести в табл.4.2.5-1.

Таблица 4.2.5-1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Объект** | **Имя объекта** | | **Свойство** | **Значение свойства** |
| **Форма** | **Form1** | | Name | *Form1* |
| Text | *Вычисление периметра и площади треугольника* |
| BackColor (Palette) | *Светло-зеленый* |
| **Метка** | **Label1** | | Text | *Введите значения сторон треугольника* |
| **Метка** | **Label2** | | Text | *A* |
| **Метка** | **Label3** | | Text | *B* |
| **Метка** | **Label4** | | Text | *C* |
| Выделить метки **Label2 – Label4**  (клавиша <Shift> + мышь) и установить одинаковые  свойства для этих меток  всех меток свойства | | | TextAlign | *2 - Center* |
| BackStyle | *0 - Transparent* |
| Font | *Arial, жирный, 12* |
| ForeColor | *Коричневыый* |
| **Метка** | | **Label5** | Text | *Периметр треугольника* |
| **Метка** | | **Label6** | Text | *Площадь треугольника* |
| Выделить метки **Label1, Label5, Label6** и установить одинаковые для всех меток свойства | | | TextAlign | *2 - Center* |
| BackColor (Palette) | *Светло-серый* |
| Font | *Arial, жирный, 12* |
| ForeColor | *Темно-зеленый* |
| Текстовое поле | | **TextBox1** | Name | *TextBox1* |
| Text | *Пусто (стереть значение, ус­тановленное "по умолчанию")* |
| Текстовое поле | | **TextBox2** | Name | *TextBox2* |
| Text | *Пусто* |
| Текстовое поле | | **TextBox3** | Name | *TextBox3* |
| Text | *Пусто* |
| Текстовое поле | | **TextBox4** | Name | *TextBox4* |
| Text | *Пусто* |
| Текстовое поле | | **TextBox5** | Name | *TextBox5* |
| Text | *Пусто* |
| Выделить текстовые поля**TextBox1- TextBox5**и установить одинаковые для всех полей свойства | | | TextBox4 | *2 – Center* |
| Font | *Arial, обычный, 12* |
| **Кнопка** | | **Button1** | Name | *Button1* |
| Text | *ВЫЧИСЛИТЬ* |
| **Кнопка** | | **Button2** | Name | *Button2* |
| Text | *ЗАВЕРШИТЬ* |
| Выделить кнопки **Button1** и**Button2** и установить одинаковое для кнопок свойство | | | Font | *Arial, жирный, 12* |

1. **Разработка схемы алгоритма**

Схему алгоритма задачи реализовать с помощью графических элементов схем алгоритмов (рис. 4.2.5-5).

* Более подробно о средствах описания алгоритмов и правилах
* оформления схем-алгоритмов можно прочитать во Введении
* пособия.

|  |
| --- |
|  |

*Рис. 4.2.5-5. Схема алгоритма вычисления периметра и площади треугольника*

1. **Написание программного кода**

Дописать программный код для обработки событий, нажав

кнопки **Button1**и **Button2***,* используя окно **Редактор кода**.

Внутри окна Редактора кода находятся строки кода, связанного с текущей формой. Операторы программы, которые используются совместно для выполнения некоторого действия, обычно группируются в программные конструкции, называемые процедурами. Обычным типом процедуры является процедура **Sub**, называемая процедурой - подпрограммой. Процедуры **Sub** в первой строке содержат ключевое слово **Sub** и заканчиваются оператором **End Sub**. Обычно процедуры выполняются при определенных обстоятельствах, например, таких, как щелчок на кнопке или вызовом специального оператора. Когда процедура ассоциирована с конкретным объектом и событием, она называется событийной процедурой.

При двойном щелчке мышью на кнопке **ЗАВЕРШИТЬ** VB автоматически добавляет первую и последнюю строки процедуры события **Button2\_Click()**.

Тело процедуры, которое находится между заголовком процедуры и оператором, завершающим процедуру – **Sub…EndSub**, будет исполняться каждый раз, когда пользователь активизирует элемент интерфейса, ассоциированный с этой процедурой. В данном случае событие - это щелчок мышью на кнопке **Button2**, но как будет показано далее, это может быть и другой тип события.

Между заголовком и концом процедуры ввести оператор **End**, а затем нажать на клавишу со стрелкой вниз.

После ввода оператора буквы становятся синими и создаются отступы, что указывает, что VB распознал **End** как одно из нескольких сотен зарезервированных или ключевых слов языка VB. Ключевое слово **End** используется для остановки программы и удаления с экрана ее формы. В данном случае **End** также является законченным оператором программы, то есть инструкцией, исполняемой компилятором VB. Напомним, что компилятор – это часть VS, обрабатывающая и выполняющая анализ каждой строки исходного кода VB, и объединяющая результаты с другими ресурсами с целью создания исполняемого файла. Все операторы программы должны следовать грамматическим правилам компилятора. В VS операторы программ могут состоять из ключевых слов, свойств, имен объектов, переменных, чисел, специальных символов и других значений. При вводе операторов программы и внесении других изменений, окно Редактор кода выполняет многие задачи, связанные с форматированием, включая настройку отступов, пробелов и добавление необходимых скобок. Точное написание оператора программы называется синтаксисом оператора.

При нажатии на клавишу «стрелка вниз» оператор **End** был сдвинут вправо так, чтобы он был отделен от операторов **Private Sub** и **End Sub**. Эта схема форматирования является одним из соглашений программирования и предназначена для того, чтобы программы выглядели ясно и читаемо. Соглашения относительно того, как организуется код в программе, часто называется стилем программирования.

Теперь, когда написан программный код, связанный с кнопкой   
**ЗАВЕРШИТЬ**, необходимо написать код для кнопки **ВЫЧИСЛИТЬ**.

Написать программный код, который будет выполняться при нажатии кнопки **ВЫЧИСЛИТЬ**, то есть событийную процедуру для кнопки **Button1**.

Чтобы отобразить форму, щелкнуть в окне **Обозреватель решений** на кнопке **Конструкто**р. В режиме **Редактора программного код**а форма, с которой производится работа, не отображается. Кнопка **Конструктор** является одним из тех механизмов, которые используются для того, чтобы снова вывести форму на экран. Также можно щелкнуть на вкладке **Form1.vb[Конструктор]**.

Дважды щелкнуть мышью на кнопке формы **ВЫЧИСЛИТ**Ь.

Через некоторое время появится окно Редактор кода, а рядом с процедурой события, связанной с кнопкой **Button2**, появится процедура события, ассоциированная с кнопкой **Button1**.

Хотя текст этой кнопки был изменен на «**ВЫЧИСЛИТ**Ь», ее имя в программе осталось **Button1**. (Имя и текст элемента интерфейса могут быть различными для удобства программирования). Каждый объект может иметь несколько связанных с ним процедур - по одной для каждого распознаваемого события.

Между операторами **Private Sub Button1\_Click()** и **End Sub** необходимо ввести строки, показанные на рис. 4.2.5-6.

После ввода каждой программной строки необходимо нажать <Enter> на клавиатуре, а для создания отступа необходимо нажать на <Тав> и следить за вводом операторов программы в точности так, как они напечатаны на рис. 4.2-5.6. (При вводе длинных строк Редактор кода будет прокручивать экран влево.) Если будет сделана ошибка, которая обычно определяется по подчеркиванию волнистой лини­ей, удалите неверную строку и введите ее снова.

При вводе кода программы VB форматирует ее текст и отображает различные части программы различными цветами, которые помогают распознавать различные элементы. В момент ввода свойства отображается список, содержащий все допустимые свойства объекта, так что вместо набора свойства вручную можно дважды щелкнуть на свойстве мышью. Если VB отображает сообщение об ошибке, то оператор, скорее всего, введен с ошибкой. Сверьте строку с текстом в книге, сделайте необходимые исправления и продолжайте набирать. Также можете удалить всю строку и начать вво­дить ее заново. Кроме того, VB может автоматически добавлять необходимый код.

Дописать программный код, добавив в него до оператора **Public Class Form1**операторы установки параметров VB:

|  |
| --- |
| **Private Sub** **Button1\_Click(ByVal** **sender** **As** **System.Object, \_**  **ByVal e As System.EventArgs) Handles Button1.Click**  **Dim A As Single, B As Single, C As Single**  **Dim P As Single, Pp As Single, S As Single**  *'Присваивание переменной В значения поля TextBox2*  **B = TextBox4.2.Text**  *'Присваивание переменной А значения поля TextBox1*  **A = TextBox1.Text**  *'Присваивание переменной С значения поля TextBox3*  **C = TextBox3.Text**  *'Вычисление периметра и присваивание значения Р*  **P = (A + B + C)**  *'Вычисление полупериметра и присваивание значения Рр*  **Pp = P / 2**  *'Вычисление площади и присваивание знач. переменной S*  **S = Sqrt(Pp \* (Pp - A) \* (Pp - B) \* (Pp - C)))**  *'Вывод знач. полупериметра в текстовое поле TextBox4*  **TextBox4.Text = P**  *'Вывод значения площади в текстовое поле TextBox5.*  **TextBox5.Text = S**  **End Sub**  **Private Sub** **Button2\_Click(ByVal sender As** **System.Object, \_**  **ByVal** e **As** **System.EventArgs) Handles Button4.2.Click**  **End**  **End Sub** |
|  |

*Рис. 4.2.5-6. Программный код процедур события*

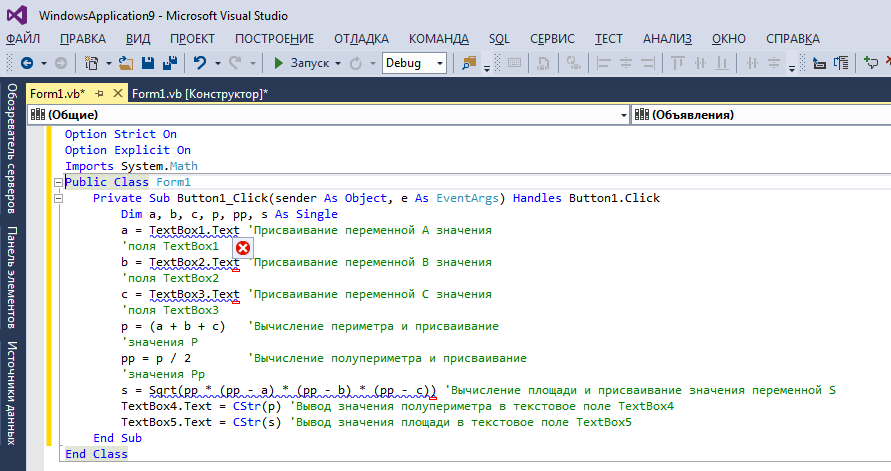
Окно панели Редактора кода будет выглядеть так, как показано на рис.4.2.5-7.

Элементы VB, отображаемые на экране зеленым цветом, называются комментарием. Комментарии – это описательные пояснения, включаемые в программный код после символа апострофа ('). Программисты используют комментарии для описания действия операторов программы. VB при запуске программы не обрабатывает эти пояснения; они существуют только для документирования того, что делает программа. При написании программы часто придется использовать комментарии, чтобы оставить легко воспринимаемые записи о том, что же вы сделали.

Сохранить введенный программный код, выполнив, например, команду ***Сохранить все*** элемента **Главного меню** элемента **Файл**.

Команда Сохранить все сохраняет все файлы, входящие в проект – собственно файл проекта, файл формы, все модули кода и все связанные с приложением компоненты. Если это первое сохранение проекта, появится диалоговое окно **Сохранить проект,** запрашивающее имя и место для сохранения проекта. (Если ваша копия VS настроена так, чтобы запрашивать место сохранения проекта при его создании, то диалогового окна **Сохранить проект** вы не увидите – VS просто сохранит все изменения.)

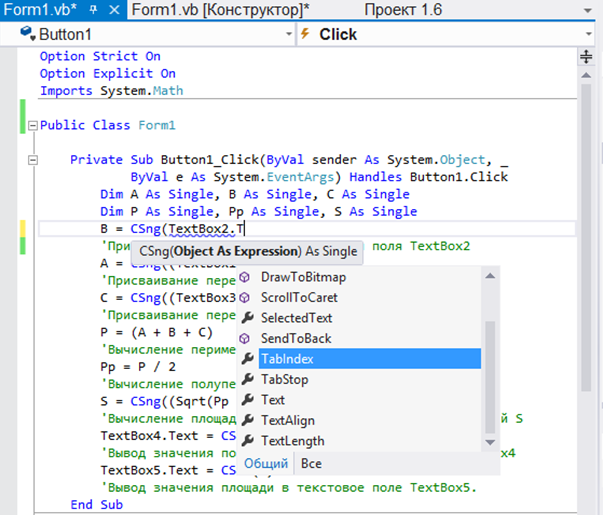
Щелкнуть на кнопке **Обзор** справа от текстового поля **Местоположение** и выбрать место для сохранения файлов.



*Рис.4.2.5-7. Окно редактора кода*

1. **Технология *IntelliSense* и точечная нотация**

Теперь нам известно, как с помощью VB читать и записывать значения свойств форм и эле­ментов управления. Если у каждой формы и элемента управления много свойств, как за ними уследить? Не слишком удобно постоянно переключаться меж­ду **Редактором кода** и окном **Свойства**, чтобы выяс­нить, какие свойства нам нужны. Имеется более удобный способ.



*Рис. 4.2.5-8. Пример использования технологии IntelliSense*

Можно использовать технологию IntelliSense. IntelliSense позволяет экономить время при редактировании программного кода в окне редактора. IntelliSense знает, к какому элементу управления вы обра­щаетесь и, какие у него есть свойства; выводит список свойств и позволяет вам выбирать из него нужное. Используемое чаще всего свойство подсвечи­вается, когда IntelliSense отображает список. Напри­мер, для текстового поля изначально подсвечивается свойство Text. С помощью стрелок вверх и вниз на клавиатуре можно просматривать список и выбирать то свойство, которое вам требуется. При нажа­тии клавиши <Tab> клавиатуры выбранное свойство добавляется в код. Используя IntelliSense, не придется запо­минать все свойства всех элементов управления. Кроме того, данное средство заметно сокращает по­требность набирать длинные имена. Например,

Во время ввода с клавиатуры текста программного кода и, в частности, **TextBox2**, нажмем клавишу «точка». Как только будет введена точка, IntelliSense отобразит список свойств элемента **TextBox2**. По умолчанию в списке IntelliSense подсвечивается чаще всего используемое свой­ство (рис. 4.2.5-8).

Если далее будет введена буква, например, буква «**T**», IntelliSense про­крутит список до первого по алфавиту свой­ства, начинающегося с «**T**». По мере ввода IntelliSense ищет в списке свойство, наибо­лее соответствующее введенным буквам.

Когда свойство подсвечи­вается в списке IntelliSense, рядом появляется под­сказка, кратко его описы­вающая. Эта подсказка, например, может сооб­щить, что свойство до­ступно только для чтения (**Read Only**, т. е. ему нель­зя присваивать значение). Кроме того, подсказка со­общает, какие значения можно присваивать свой­ству, например, строку (текст) или целое число.

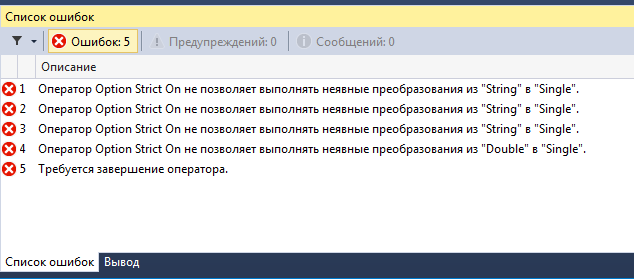
1. **Отладка и внесение изменений в программу**

На этом этапе проверяется правильность работы программы. Ошибки, возникающие в процессе создания программы, могут быть вызваны и неадекватным моделированием, и некорректностью метода или алгоритма, и, наконец, неправильным применением самих средств VB.

В целом типы ошибок принято разделять на два неравнозначных класса. Один из них – это класс синтаксических ошибок, то есть ошибок, связанных с неправильной записью или употреблением языковых конструкций. Эти ошибки легко исправимы, так как соответствующее программное обеспечение – транслятор – осуществляет автоматический контроль синтаксической правильности программ пользователя, а с помощью контекстно-зависимой помощи можно получить как разъяснения об ошибке, так и узнать правильный вид языковой конструкции.

Другой вид ошибок, действительно представляющий проблему программирования, – смысловые ошибки. Обнаружение и исправление их, что собственно и представляет собой процесс отладки, дело очень сложное:

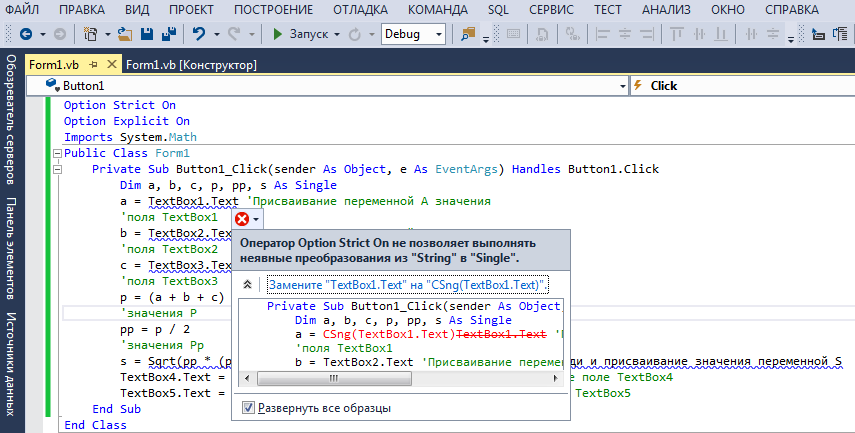
* + Как определить, что программа имеет смысловую ошибку? В лучшем случае программа не работает, то есть её работа прерывается в некоторый момент, и система выдаёт какое-нибудь сообщение типа «исчезновение порядка числа с плавающей точкой». В худшем случае программа успешно завершает свою работу и выдаёт результаты, отвечающие интуитивным представлениям о характере решения задачи.
  + Как обнаружить скрытые ошибки? Самый популярный метод – тестирование программы. Следует взять такие исходные данные, правильный результат расчёта для которых известен заранее, и выполнить программу с этими данными. Если полученный результат совпадает с известным результатом, то, как говорят, «тест прошёл успешно». Неприятность заключается в том, что, это совсем не означает, что программа не содержит ошибок.
  + Запустить программу, приведенную на рис. 4.2.5-7, на выполнение. Список полученных ошибок приведен на   
    рис. 4.2.5-9.



*Рис.4.2.5.9. Список ошибок*

Так как в программе запрещено использовать неявные преобразования (оператор **Option Strict** имеет положение On), то преобразования из **String** в **Single** и из **Double** в **Single** невозможны.

Это было ясно и без выполнения программного кода, поскольку технология **IntelliSense** не только указывает на ошибку в преобразовании (красный прямоугольник на ошибочной строке), но и помогает исправить ошибку (рис. 4.2.5-10).



*Рис. 4.2.5-10. Иллюстрация использования технологии IntelliSense*

Подвести курсор на красный прямоугольник и щелкнуть на строке появившегося квадрата. Появится текст с параметрами исправления ошибок и их автоматического исправления.

VB считает содержимое текстового поля текстом (**String**), а не числом (**Single**), даже если в текстовое поле вводятся цифры. Поэтому для присваивания числовым переменным **A, B** и **C** значений, введенных в текстовые поля объектов **TextBox1**, **TextBox2** и **TextBox3**, необходимо предварительно осуществить преобразование данных из типа **String** в тип **Single**. За такое преобразование отвечает функция **CSng(**).

Аналогично при выводе вычисленных числовых значений периметра и площади в текстовые поля **TextBox4** и **TextBox5** числовые значения предварительно преобразуются в текст (**String**) с помощью функции **CStr().**

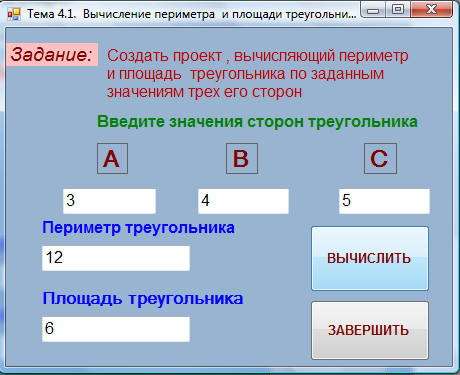
Последняя пятая ошибка – в выражении стоит лишняя скобка.

После устранения ошибок (рис. 4.2.5-11) необходимо запустить программу на выполнение и ввести значения сторон треугольника, как показано на рис. 4.2.5-12.

|  |
| --- |
| **Option Strict On**  **Option Explicit On**  **Imports System.Math**  **Public Class Form1**  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles Button1.Click**  **Dim A, B, C As Single**  **Dim P, Pp, S As Single**  *'Присваивание переменной В значения поля TextBox2*  **B = CSng(TextBox4.2.Text)**  *'Присваивание переменной А значения поля TextBox1*  **A = CSng((TextBox1.Text))**  *'Присваивание переменной С значения поля TextBox3*  **C = CSng((TextBox3.Text))**  *'Вычисление периметра и присваивание значения Р*  **P = (A + B + C)**  *'Вычисление полупериметра и присваивание значения Рр*  **Pp = P / 2**  *'Вычисление площади и присваивание значения переменной S*  **S = CSng((Sqrt(Pp \* (Pp - A) \* (Pp - B) \* (Pp - C))))**  *'Вывод значения полупериметра в текстовое поле TextBox4*  **TextBox4.Text = CStr(P)**  *'Вывод значения площади в текстовое поле TextBox5.*  **TextBox5.Text = CStr(S)**  **End Sub**  **Private Sub Button2\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles Button4.2.Click**  **End**  **End Sub**  **End Class** |
|  |

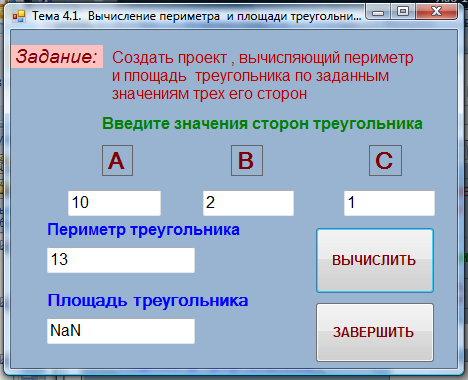
*Рис.4.2.5-11 Усовершенствованный программный код проекта*

* Далее необходимо проанализировать результаты вычислений. Очевидно, что при тех данных, которые были введены, периметр треугольника должен быть равен 12, а площадь – 6. Отсутствие сообщений компьютера о синтаксических ошибках есть необходимое, но недостаточное условие правильности программы.
* Поскольку программа достаточна простая, то полученный результат можно проверить вручную, а именно запустить программу на выполнение еще раз и ввести значения сторон, как показано на рис. 4.2.5-13. Система выдаст результат с ошибкой в поле Площадь треугольника.



*Рис.4.2.5-12. Результат выполнения проекта с исходными данными:*

*А=3, В=4, С=5*



*Рис.4.2.5-13 Результат выполнения проекта с исходнымиданными: А=10, В=2, С=1*

Ошибка произошла в связи с тем, что были введены данные, при которых треугольник не может существовать: подкоренное выражение имеет отрицательное значение. Обратите внимание, что все программы должны проверять вводимые данные на корректность. Проверить работу кнопки ЗАВЕРШИТЬ. Щелчок по ней должен привести к завершению программы. Сохраните изменения в проекте.

1. **Пошаговое выполнение программы**

**Контроль программы до ввода**. Чтобы ускорить процесс создания проекта, рекомендуется проконтролировать программу еще до ввода в компьютер с помощью просмотра, проверки и прокрутки.

Просмотр текста программы предусматривает обнаружение описок и расхождений с алгоритмом. При проверке программы разработчик программного кода по тексту программы мысленно воспроизводит тот вычислительный процесс, который определяет программа, после чего сверяет его с требуемым процессом.

Основой прокрутки является имитация выполнения программы. Для выполнения прокрутки используют простейшие исходные данные и над ними производят все необходимые вычисления, следуя тексту программы.

Следующим этапом контроля правильности программы является отладка и тестирование на компьютере.

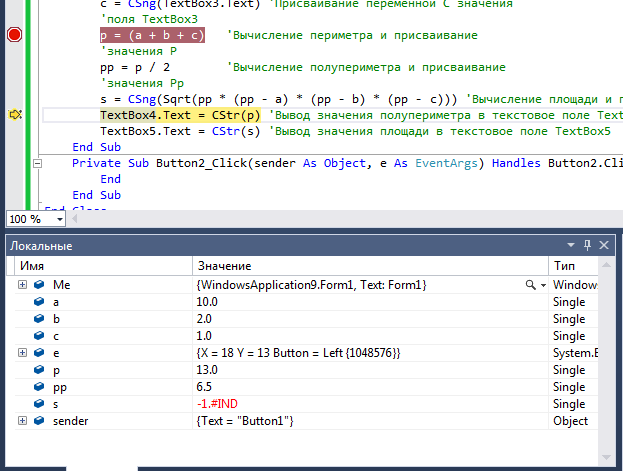
**Отладка программы** – это процесс поиска и устранения ошибок в программе, производимый по результатам её прогона на компьютере, а тестирование – это испытание, проверка правильности работы программы в целом, либо её составных частей.

Отладка и тестирование – это два четко различимых и непохожих друг на друга этапа, поскольку при отладке происходит локализация и устранение синтаксических ошибок и явных ошибок кодирования, а в процессе тестирования проверяется работоспособность программы, не содержащей явных ошибок. Таким образом, тестирование устанавливает факт наличия ошибок, а отладка выясняет ее причину.

В современных программных системах отладка осуществляется часто с использованием специальных программных средств, называемых отладчиками. Программа-отладчик обычно обеспечивает следующие возможности:

* пошаговое исполнение программы с остановкой после каждой команды (оператора);
* просмотр текущего значения любой переменной или нахождение значения любого выражения, в том числе, с использованием стандартных функций; при необходимости можно установить новое значение переменной;
* установку в программе «точек останова», т.е. точек, в которых программа временно прекращает свое выполнение, так что можно оценить промежуточные результаты, и др.

**Пошаговое выполнение**. Выполним пошаговое выполнение программы, с исходными данными показанными на рис. 4.2.5-13, для того чтобы выявить проблему возникновения ошибки **NaN**. Поставим, так называемую точку останова после ввода исходных данных (перед оператором **p=(a+b+c)**) и нажмем клавишу <F9>, в результате чего строка оператора выделяется красным цветом. Теперь после ввода исходных данных в текстовые поля и нажатия на кнопку **Вычислить** выполнение программы прерывается перед вычислением выражения.



*Рис.4.2.5-14. Иллюстрация работы Отладчика*

Открывается окно редактора кода и под ним окно Локальные, в котором отображаются значения всех объектов, использующихся в данном программном коде. После каждого нажатия клавишу <F10>, будет происходить пошаговое выполнение программы и изменение значений всех переменных.

Из рис. 4.2.5-14 видно, что подкоренное выражение, а, следовательно, значение ячейки **s** равно отрицательному значению, поэтому возникает ошибка в поле вывода **s**.

Чтобы стереть точку останова для последующих запусков программы, надо в элементе меню Отладка выполнить команду   
***Удалить все точки останова*** или щелкнуть мышью на красном кружке (точке останова) слева от оператора, на котором прерывается программа.

1. **Создание выполняемого (.exe) файла**

Окончательная компоновка проекта состоит в создании .exe файла:

Проект сохранить.

Из элемента Главного меню Построение выполнить команду *Построить Имя проекта*.

Созданный .exe файл можно найти в текущем каталоге проекта, например, …/Запись выражений /папка Bin /папка Release.

1. **Выход из Visual Studio**

Сохранить все изменения, которые были сделаны.

Для этого щелкнуть на кнопке *Сохранить все* на Стандартной панели инструментов.

По умолчанию VSприсвоит имя программе при создании проекта, но при этом не указывает местоположение файла и проект не сохраняется до тех пор, пока вы не щелкнуть на кнопке ***Сохранить все***, или не выполните команду ***Сохранить все*** из элемента Главного меню – **Файл.**

В меню **Фай**л выполнить команду **Ex**it.

Visual Studio закроется**.**

Таким образом, мы полностью выполнили этапы создания проекта среде VS, очень подробно рассмотрев все тонкости создания проекта, начиная с постановки задачи и заканчивая созданием исполняемого файла. Создание других проектов в учебном пособии будет описано менее подробно, поскольку читатель может всегда обратиться к вышерассмотренному приложению.

### 4.2.6. Тестовые задания

1. Синтаксис языка программирования – это
2. набор правил истолкования отдельных языковых конструкций
3. грамматические правила языка
4. правила компиляции операторов языка
5. Алфавит языка программирования – это
6. переменные и операторы языка
7. множество символов, из которых строятся все конструкции языка
8. множество идентификаторов
9. Идентификаторы – это
10. имена лексических элементов языка
11. имена переменных
12. ключевые слова
13. имена операторов
14. Ключевые слова – это
15. идентификаторы, зарезервированные для специального использования
16. операторы языка
17. константы языка
18. операторы языка
19. Для ввода данных в схемах алгоритма используют фигуру
20. ромб
21. прямоугольник
22. треугольник
23. параллелограмм
24. Внутреннее представление данных в памяти компьютера

определяется

1. типом данных
2. системой счисления
3. системой программирования
4. Последовательность символов, заключенная в двойные кавычки, называется
5. строковой переменной
6. строковой константой
7. числовой переменной
8. числовой константой
9. Величина, не изменяющая своего значения в ходе выполнения программы - это
10. функция
11. переменная
12. константа
13. оператор
14. Величина, к которой обращаются по имени, принимающая различные значения в ходе выполнения программы, называется
15. строкой
16. символом
17. константой
18. переменной
19. Показательная функция записывается как
20. Exp(x)
21. Abs(x)
22. Sqrt(x)
23. Atan(x)
24. Из перечисленных действий наивысший приоритет имеет
25. сложение
26. возведение в степень
27. вычитание
28. умножение
29. В качестве аргумента встроенной математической функции можно применять
30. число
31. любое арифметическое выражение
32. переменные
33. все ответы верные
34. Операция Mod определяет
35. модуль числа
36. квадратный корень числа
37. знак числа
38. остаток от деления
39. Числовые переменные могут быть
40. целыми и вещественными
41. целыми и символьными
42. простыми и сложными
43. нет верного ответа
44. Наибольшее целое число, не превосходящее аргумент, возвращает функция
45. Int(Х)
46. Fix(X)
47. CInt(X)
48. Sign(X)
49. Ошибки, связанные с неправильной записью конструкций языка программирования, называются
50. логическими
51. орфографическими
52. математическими
53. синтаксическими
54. Запись TextBox1.Text=CStr(у) предназначена
55. для вывода в текстовое поле формы значения переменной у
56. для ввода данных с клавиатуры
57. для вывода числовых данных
58. для печати результатов на принтере
59. Оператор присваивания изменяет
60. имя переменной
61. значение переменной
62. тип переменной
63. значение и тип одновременно
64. Синтаксически неправильная запись
65. X = Y \* Sin( X) ^2 + 4
66. X = Y \* Sin ^ 2 (X) + 4
67. X = Y \* Sin( X ^ 2 ) + 4
68. X = Y ^2 \* Sin( X) + 4
69. Оператор организации ввода данных с клавиатуры в системе программирования VB записывается с использованием ключевого слова
70. Loop
71. Inputbox
72. Print
73. Msgbox
74. Ошибки, связанные с неправильной записью конструкций языка  
    программирования, называются
75. логическими
76. орфографическими
77. математическими
78. синтаксическими
79. методическими
80. Ошибкой при программировании является
81. глупые действия, совершаемые неопытным пользователем
82. все, что заставляет программу работать не так, как нужно, или вообще не работать
83. получение результата, которого вы не ожидали
84. в Visual Basic не может быть ошибок, он сам их исправляет
85. Отсутствие сообщений компьютера о синтаксических ошибках
86. есть необходимое и достаточное условие правильности программы
87. есть достаточное условие правильности программы
88. есть необходимое, но недостаточное условие правильности программы
89. не является необходимым условием правильности программы
90. гарантирует правильность программы
91. Аргумент для функции Sin( ) задается
92. в радианах
93. в градусах
94. в секундах
95. в любых единицах
96. Тип переменной alfa, объявленный в Dim alfa AS Integer будет
97. целым
98. вещественным
99. неопределенным
100. нет верного ответа
101. Ввод данных в языке Visual Basic осуществляется с помощью
102. InputBox
103. Print
104. If
105. Do

### 4.2.8. Лабораторная работа по теме «Основные средства языка программирования VB .NET. Типы данных»

**Цель** выполнения лабораторной работы состоит в изучении алфавита языка, типов данных и правил записи арифметических выражений, содержащих константы, переменные, различные арифметические действия и «встроенные» функции, ознакомлении с основными правилами преобразования типов.

### Вопросы, подлежащие изучению

1. Алфавит языка VB.
2. Основные понятия: программа, программная строка, оператор присваивания, комментарий.
3. Данные, типы данных. Числовые константы и переменные.
4. Арифметические действия над данными.
5. «Встроенные» функции.
6. Последовательность выполнения действий в арифметических выражениях. Понятие приоритета.
7. Явные и неявные преобразования.
8. Функции ввода и вывода.

### Общее задание на разработку проекта

1. ***Изучите основные средства языка программирования VB и типы данных*** *(****Тема 4.2****).*
2. ***Выберите индивидуальный вариант*** *задания из табл.4.2.8-1.*
3. ***Разработайте графический интерфейс*** *пользователя.*
4. **Решите задачу вычисления арифметических выражений** в двух вариантах (два проекта) в одном решении: первый проект с неявными преобразованиями типов данных, второй – с требованием явного преобразования (без инструкции **Option Strict On** и с использованием этой инструкции). Для этого, формализуйте арифметические выражения, т.е. запишите выражение в формальном математическом виде, а затем запишите полученные математические формулы в виде выражения по правилам языка программирования с учетом приоритета операций.
5. Выполните созданные проекты.
6. **Выполните созданные проекты по шагам** с помощью отладчика, поставив точку останова после ввода данных.
7. ***Получите результаты****.*
8. ***Докажите правильность результата****.*

### Варианты индивидуальных заданий

Таблица 4.2.8-1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Формулы для вычислений** | | **Исходные данные** | |
| **x** | **y** |
| **1)** |  | i=t  j= | 3,59 | 17,53 |
| **2)** |  | m=d  n= | 1,674 | -0,533 |
| **3)** |  | k=h  n= | 32,01 | -0,4917 |
| **4)** |  | i=c  j= | 2,123 | -1,89 |
| **5)** |  | i=b  j= | 8,402 | -0,22226 |
| **6)** |  | m=d  n= | 1,002 | -0,5321 |
| **7)** |  | m=z  n= | 12,003 | -5,408 |
| **8)** |  | k=d  l= | 1,3802 | -1,9 |
| **9)** |  | i=f  j= | 12,678 | 6,9 |
| **10)** |  | i=a  j= | 82,578 | 1,4517 |
| **11)** |  | m = d  n = [d] | -4.62 | -0.32 |
| **12)** |  | i = f  j = [f] | 5.305, | -0.084 |
| **13)** |  | k = h  n = [h] | 0.68 | -0.848 |
| **14)** |  | i =a  j = [a] | 14.2.43 | -0.2 |
| **15)** |  | k = с  i = [с] | -1.462 | 0.577 |
| **16)** |  | i = b  j = [b] | 0.501 | 1.7 |
| **17)** |  | i = d  j = [d] | -0.45 | 24.3 |
| **18)** |  | i = f  j = [f] | 8.8 | 5.8 |
| **19)** |  | k = h  l = [h] | 0.6485 | 3.9 |
| **20)** |  | m = a  n = [a] | 125.04 | 5.55 |
| **21)** |  | k = b  l = [b] | 7.302 | 5.782 |
| **22)** |  | i = c  j = [c] | -3.356 | 0.049 |
| **23)** |  | m = a  n = [a] | 1.892 | -0.61 |
| **24)** |  | m = a  n = [a] | 45.892 | 0.601 |
| **25** |  | k = c  l = [c] | 13 | 0.54321 |
| **26)** |  | m =d  n = [d] | 4213 | 104.2.654 |
| **27)** |  | m =f  n = [f] | 29.999 | 165.091 |
| **28)** |  | k = g  I = [g] | -9.20 | -5,892 |
| **29)** |  | m = z  n = [z] | 4.51 | -0.25 |
| **30)** |  | k =b  l = [b] | 27 | 224.2.1 |

### Содержание отчета

1. Тема и название лабораторной работы.
2. Фамилия, имя студента, номер группы, номер варианта.
3. Задание на разработку проектов и вариант задания.
4. Формализация и уточнение задания: запись арифметических выражений по правилам VB, соответствующих заданным математическим формулам.
5. Элементы разрабатываемых проектов в одном решении:
6. графический интерфейс пользователя;
7. таблица свойств объектов;
8. схема алгоритма решаемой задачи;
9. программный код первого проекта;
10. программный код второго проекта.
11. Результаты выполнения проектов.
12. Доказательство правильности работы программы.

### Пример выполнения задания

1. Тема и название работы:

Основные средства языка программирования VB .NET. Типы данных.

Вычисление арифметических выражений.

1. Фамилия, имя студента, номер группы, номер варианта:

Иванов И., БИН1405, вариант 13.

1. Задание на разработку проекта и вариант задания:

Решите задачу вычисления арифметических выражений путем создания двух проектов в одном решении с именем **Проект 2.1**: первый проект – с неявными преобразованиями типов данных; второй – **Проект 2.2** с требованием явного преобразования (без инструкции **Option Strict On** и с использованием этой инструкции):



1. Формализация и уточнение задания:

Для формализации и уточнения задания допустим, что **x, y** – исходные данные вещественного типа **Double**, а переменные k, m, n, z, p – результаты вычислений, причем **k**– переменная целого

типа **Integer, m, n, z** – вещественного типа **Double, p** – вещественного типа Single. Выражение **m=[z]** означает, что вычисляется целая часть

числа, полученная с помощью функции **Fix(**), а значение переменной **n** вычисляется с помощью функции **Int()**. Кроме того, в этой задаче

дополнительно будем использовать две строковые переменные **s**1 и **s2** типа **Strin**g, которые участвуют в выражениях **s1=z** и **s2=p**.

Запишите арифметическое выражение на языке программирования:

z = (0.002 – Exp(x^y)) / ((100 – y)\*(x + 2));

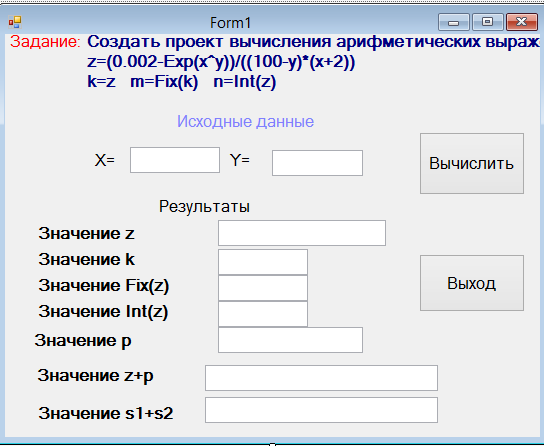
k = z; m = Fix(z).

1. Элементы разрабатываемых проектов в одном решении:
2. Графический интерфейс пользователя:

При создании проектов и решения задайте им разные имена. Например, имена проектов – **Проект 2.1** и **Проект 2.2**, а имя решения – **Лаб.4.2**. Анализируя задание на разработку проектов, можно сделать вывод о том, что для организации ввода и вывода данных можно использовать три элемента управления.

Текстовое поле **TextBox** используется как для ввода исходных данных в программу, так и для отображения результатов, получаемых в этой программе. Элемент **Label** используется для размещения на форме поясняющих надписей. Кнопки **Button** используются для запуска и завершения работы программы.

Таким образом, форма может иметь вид, показанный на   
рис.4.2.8-1.



*Рис. 4.2.8-1. Форма проектов* **Проект 2.1**и **Проект 2.2**

1. Таблица свойств объектов:

Установите и сведите в табл. 4.2.8-2 свойства всех объектов.

Таблица 4.2.8-2

|  |  |  |  |
| --- | --- | --- | --- |
| **Объект** | **Имя объекта** | **Свойство** | **Значение свойства** |
| Форма | **Form1** | Name | *Form1* |
| Text | *Вычисление арифметических выражений* |
| Метка | **Label1** | Text | *Задание: Создать проект вычисления арифметическх выражений* |
| ForeColor | *Red* |
| ImageAlign | *MiddleCenter* |
| Метка | **Label2** | Text | *Исходные данные* |
| ForeColor | *YellowGreen* |
| ImageAlign | *MiddleCenter* |
| Метка | **Label3** | Text | *Х=* |
| Метка | **Label4** | Text | *Y=* |
| Метка | **Label5** | Text | *Результаты* |
| Метка | **Label6** | Text | *Значение z* |
| Метка | **Label7** | Text | *Значение к* |
| Метка | **Label8** | Text | *Значение (Fix(z))* |
| Метка | **Label9** | Text | *Значение Int(z)* |
| Метка | **Label10** | Text | *Значение p* |
| Метка | **Label11** | Text | *Значение z+p* |
| Метка | **Label12** | Text | *Значение s1+s2* |
| Текстовое поле | **TextBox1** | Name | *TextBox1* |
| Текстовое поле | **TextBox2** | Name | *TextBox2* |
| Текстовое поле | **TextBox3** | Name | *TextBox3* |
| Текстовое поле | **TextBox4** | Name | *TextBox4* |
| Текстовое поле | **TextBox5** | Name | *TextBox5* |
| Текстовое поле | **TextBox6** | Name | *TextBox6* |
| Текстовое поле | **TextBox7** | Name | *TextBox7* |
| Текстовое поле | **TextBox8** | Name | *TextBox8* |
| Текстовое поле | **TextBox9** | Name | *TextBox9* |
| Кнопка | **Button2** | Name | *Button2* |
| Text | *Вычислить* |
| Кнопка | **Button1** | Name | *Button1* |
| Text | *Выход* |
| Кнопки | **Button1 и**  **Button2** | Font | *Arial, жирный, 12* |

1. Схема алгоритма решаемой задачи:

Схема алгоритма представлена на рис. 4.2.8-2.



*Рис. 4.2.8-2.Схема алгоритма вычисления арифметических выражений*

1. Программный код первого проекта:

В вычисляемом выражении используется математическая

функция **Exp()**, поэтому требуется подключить системную

библиотеку **Math**.

Кроме того, две функции **Fix()** и **Int()**, которые выделяют из вещественного числа его целую часть (по-разному: для положительных и отрицательных чисел), но не преобразуют результат в целый тип, он остается вещественным числом. Также следует обратить внимание, что операция сложения «+» по-разному работает разных типов данных.

Разделителем целой и дробной частей вводимого вещественного числа обязательно должна быть запятая, а не точка.

Код программы первого проекта (**Проект 2.1**) приведен на   
рис. 4.2.8-3.

|  |
| --- |
| **Imports System.Math***'подключение математического модуля*  **Public Class Form1**  **Private Sub Button1\_Click(sender As Object, e As EventArgs)\_**  **Handles Button1.Click**  **Dim х, y, z, m, n As Double, p As Single, k As Integer**  **Dim s1, s2 As String**  **х = TextBox1.Text**  **y = TextBox4.2.Text**  **z = (0.002 - Exp(х ^ y)) / ((100 - y) \* (х + 2))**  **p = z**  **k = z**  **m = Fix(z)**  **n = Int(z)**  **s1 = z**  **s2 = p**  **TextBox3.Text = z**  **TextBox4.Text = k**  **TextBox5.Text = m**  **TextBox6.Text = n**  **TextBox7.Text = p**  **TextBox8.Text = z + p**  **TextBox9.Text = s1 + s2**  **End Sub**  **Private Sub Button2\_Click(sender As Object, e As EventArgs) \_**  **Handles Button4.2.Click**  **End**  **End Sub**  **End Class** |

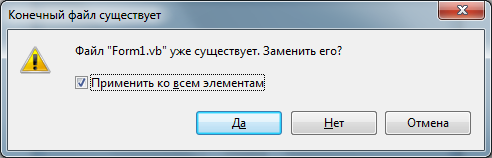
*Рис. 4.2.8-3. Программный код проекта* **Проект 2.1**

1. Программный код второго проекта:

Создайте второй проект с именем **Проект 2.2** в уже имеющемся решении. Для этого надо выполнить команду **Файл/Добавить/Создать проект/Приложение Windows Forms**. В открывшемся окне **Добавить новый проект** задайте имя проекта **Проект 2.2**.

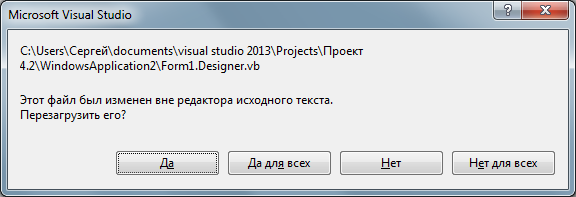
Чтобы заново не создавать существующий интерфейс и не копировать программный код, в окне **Обозреватель решений** нажмите правую кнопку мыши на элементе **Проект 2.2** и выберете пункт **Существующий элемент**. Далее в открывшемся окне **Добавление существующего элемента** найдите файл **Form1.vb** первого проекта с именем **Проект 2.1** и нажмите кнопку **Добавит**ь.

В открывшемся окне (рис. 4.2.8-4) поставьте флажок в опции **Применить ко всем элементам** и нажмите кнопку Да.



*Рис. 4.2.8-4. Диалоговое окно VS*

Затем в окне (см. рис4.2.8-5) выберите кнопку Да для всех.



*Рис.* 4.2.8*-5. Диалоговое окно VS*

1. **Выполнение проектов и их результаты.**

По умолчанию компилятор VB.NET устанавливает режим явного объявления переменных. Таким образом, в программе непременно должен присутствовать оператор описания переменных **Dim**.

Ниже приведены описания всех трех режимов:

**Option Explicit**. Если задан режим **Option Explicit**, то каждая переменная в проекте должна быть объявлена. Эта установка не будет играть роли при использовании режима **Option Strict**, поскольку компилятор не сможет определить тип необъявленной переменной. Однако режим **Option Explicit** никогда не следует отключать.

**Option Strict**. При включенном режиме **Option Strict On** компилятор должен иметь возможность определить тип каждой переменной. Кроме того, если одной переменной присваивается значение другой и при этом необходимо преобразование типов (например, от **Integer** к **Boolean**), такое преобразование должно быть явным.

**Option Compare**. Данный режим определяет, будут ли сравниваться строки как наборы двоичных чисел (**Binary**) или как массивы символов в текстовом формате (Text).

**Выполните первый проект с помощью отладчика по шагам**. Поставьте точку останова после ввода исходных данных (**x** и **y**) в текстовые поля на операторе, вычисляющем арифметическое выражение (**z = …**).

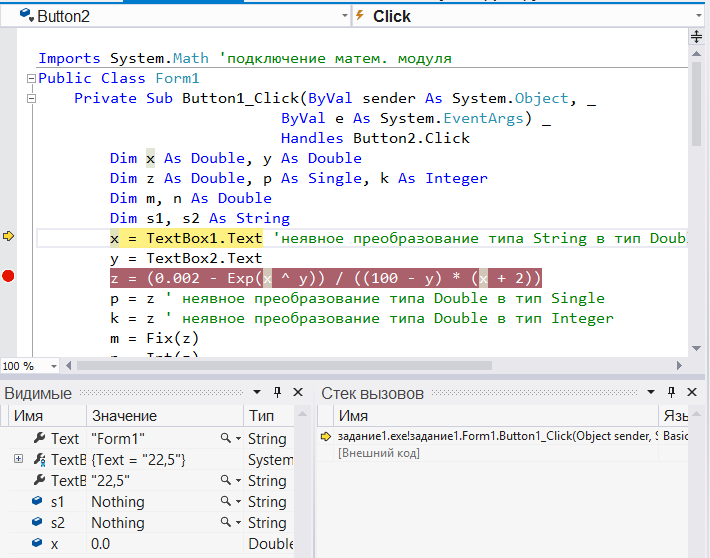
Для этого следует подвести курсор к этому оператору и нажать клавишу <F9> клавиатуры, в результате чего строка оператора выделяется красным цветом. Теперь после ввода исходных данных в текстовые поля и нажатия на кнопку **Вычислить** выполнение программы прерывается перед вычислением выражения. Открывается окно редактора кода и под ним окно **Видимые**, в котором отображаются значения всех объектов, использующихся в данном программном коде. Для выполнения следующего оператора нужно нажать клавишу <F10>, в результате чего

в окне изменяется значение переменной **z**, которая выделяется красным цветом. Нажимая клавишу <F10>, происходит пошаговое выполнение программы. Проследите изменение значений всех переменных.

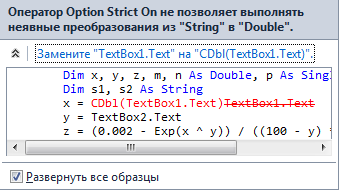
Чтобы стереть точку останова для последующих запусков программы, выполните команду **Удалить все точки останова** элемента основного меню **Отладка** или щёлкните мышью на красном кружке (точке останова) слева от оператора, на котором прерывается программа.

Посмотрите, какие изменения произойдут в программе при включенном режиме **Option Strict**(второй проект). Выставите его в положение On и запустите программу на выполнение.

Теперь проект перестал правильно работать, красным цветом выделены отдельные строки (рис. 4.2.8-6).



*Рис. 4.2.8-6. Окно***Редактора программного кода** *и окно* **Видимые**



*Рис. 4.2.8-7. Окно VS, показывающее на ошибки*

*в программном коде и пути их исправления*

Подведя указатель мыши на строку **x=TextBox1.Text**, можно прочитать, что **Оператор Option Strict On не позволяет выполнять неявные преобразования** из типа **String** в **Double**. Действительно, текстовое окно изначально служит для ввода текста, а мы пытаемся ввести число вещественного типа в переменную **х** типа **Double**. Это несоответствие вызывает появление ошибки. Другие несоответствия можно посмотреть самостоятельно, подводя указатель мыши на красные прямоугольники строк.

Интеллектуальная среда сама показывает, какие изменения следует сделать в строке программного кода, для этого щелкните на красном крестике в конце строки, где произошла ошибка (рис. П.1.2-7).

|  |
| --- |
| **Option Strict On**  **Imports System.Math***'подключение математического модуля*  **Public Class Form1**  **Private Sub Button1\_Click(sender As Object, \_**  **e As EventArgs) Handles Button1.Click**  **Dim х, y, z, m, n As Double, p As Single, k As Integer**  **Dim s1, s2 As String**  **х = CDbl(TextBox1.Text)***'преобразование типа String в Double*  **y = CDbl(TextBox4.2.Text)**  **z = (0.002 - Exp(х ^ y)) / ((100 - y) \* (х + 2))**  **p = CSng(z)** *'преобразование типа Double в тип Single*  **k = CInt(z)** *'преобразование типа Double в тип Integer*  **m = Fix(z) : n = Int(z)**  **s1 = CStr(z) : s2 = CStr(p)**  *'преобразование числовых типов в String*  **TextBox3.Text = CStr(z) : TextBox4.Text = CStr(k)**  **TextBox5.Text = CStr(m) : TextBox6.Text = CStr(n)**  **TextBox7.Text = CStr(p)**  **TextBox8.Text = CStr(z + p)** *'сложение чисел*  **TextBox9.Text = s1 + *s2*** *'сложение строк*  **End Sub**  **Private Sub Button2\_Click(sender As Object, \_**  **e As EventArgs) Handles Button4.2.Click**  **End**  **End Sub**  **EndClass** |

*Рис. 4.2.8-8. Исправленный программный код* ***Проект 2.1***

Действительно, данные, введенные в текстовое поле, являются строкой, а не числом, поэтому для корректных вычислений необходимо использовать явное преобразование текстового аргумента в числовое значение с помощью **CDbl()**. Функция **CStr()** выполняет обратное преобразование числового значения в строковый тип. Для получения результата целого типа (переменной **k**) из вещественной переменной **z** используется функция **CInt()**, которая преобразует значение этой переменной в целое число, округляя его по правилам математики. Обратите внимание, что режим **Option Strict On** можно включить вручную, указав данную строку в начале программного кода**.**

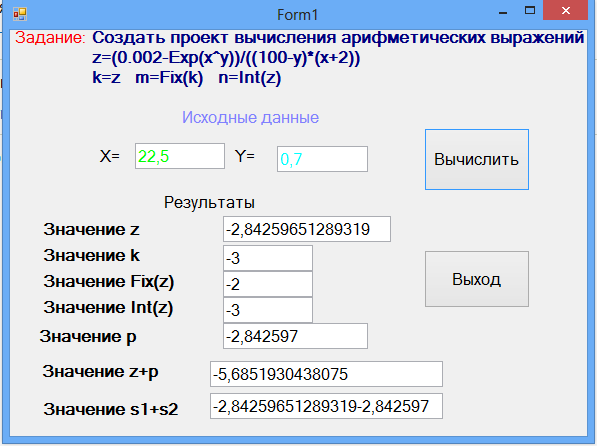
Окончательный код программы приведен на рис. П.1.2-8

Выполните второй проект.

Для этого второй проект с именем **Проект 2.2** надо назначить запускаемым проектом. Выделите в окне **Обозреватель решений** имя проекта **Проект 2.2** и, нажав правую кнопку мыши, выберете пункт ***Назначить запускаемым проектом***.

**Результаты выполнения проектов.**

Результат разработанных проектов представлен на рис. *4.2.8*-9.



*Рис. 4.2.8-9. Результат работы* Проекта 2.1 *и* Проекта 2.2

1. **Доказательство правильности работы программы.**

Для доказательства правильности результатов выполните расчет арифметического выражения с использованием калькулятора или программы Microsoft Excel, сделайте вывод о том, что результаты, полученные вручную, подтверждают правильность выполнения программы.

### Контрольные вопросы

1. Что такое семантика языка программирования?
2. Что такое синтаксис языка программирования?
3. Что такое алфавит языка программирования?
4. Что такое идентификаторы?
5. Что такое ключевые слова?
6. Какие операторы языка программирования известны?
7. Чем определяется внутреннее представление данных в памяти компьютера?
8. Как называется последовательность символов, заключенная в кавычки?
9. Как называется величина, не изменяющая своего значения в ходе выполнения программы?
10. Какие действия выполняет оператор присваивания?
11. Что необходимо включить в программный код для доступа к математическим встроенным функциям?
12. Какая функция вычисляет квадратный корень числа?
13. Какие ошибки встречаются при записи числовой переменной?
14. Как выглядит правильная запись числовой переменной в языке VB

.NET?

1. Как называется величина, к которой обращаются по имени, принимающая различные значения в ходе выполнения программы?
2. Как выглядит правильная запись символьной переменной?
3. Что такое выражение?
4. Что может входить в арифметическое выражение?
5. Какие бывают типы переменных?
6. Что такое тип данных?
7. Какие встроенные стандартные функции вы знаете?
8. Что используется в качестве аргумента стандартной функции?
9. Что вычисляет стандартная функция **Sqrt(x)**?
10. Что вычисляет стандартная функция **Fix(x)**?
11. Что вычисляет стандартная функция **Sign(x)**?
12. Какие типы могут иметь числовые переменные?
13. Какая функция возвращает аргумент, округленный до целого по

правилам округления математики?

1. Как называются ошибки, связанные с неправильной записью

конструкцийязыкапрограммирования?

1. О чем говорит отсутствие сообщений компьютера о синтаксическихошибках?
2. В каких единицах задается аргумент для функции **Sin(x)**?
3. Каким образом осуществляется ввод исходных данных?
4. О чем говорит запись **TextBox1.Text = у**?
5. Что изменяет операция присваивания?
6. Какой тип данных используется для объявления символьных строк?
7. Какой тип данных используется для объявления переменных,

принимающих целое значение?

1. Какие функции преобразуют строку символов в числовое значение?
2. Какие функции преобразуют числовое значение в символьную строку?
3. Чем определяется правильный порядок выполнения операций в

выражении?

1. Какой тип имеет значение текстового поля в форме?
2. В каких случаях в программе должны присутствовать явные

преобразования типов?

1. Чем разделяются операторы в программе, записанные на одной строке?
2. Что необходимо для переноса длинных строк программного кода?
3. Для чего код программы нужно сопровождать комментариями?
4. Что представляет собой проект?
5. Как размещаются элементы управления на форме?
6. Как связываются элемент управления и программный код?
7. Каковы правила записи идентификаторов и выражений?
8. Из чего состоит программа на языке VB?
9. Что представляют собой понятия: программа, программная строка,

оператор, комментарий?

1. В чем отличие константы от переменной?