МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра информационной безопасности

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г. Локтионова
«11» ноября 2017г.

ОТЛАДКА ПРОГРАММ И ОБРАБОТКА ОШИБОК

Методические указания по выполнению лабораторной работы по дисциплинам «Основы реверсинжиниринга программных средств», «Методы защиты программного обеспечения» для студентов специальности 10.03.01

Курск 2017

УДК 004.056.55

Составитель А.Л. Марухленко

Рецензент
Кандидат технических наук, доцент *И.В. Калуцкий*

Отладка программ и обработка ошибок: методические указания по выполнению лабораторных работ по дисциплине «Основы реверсинжениринга программных средств», «Методы защиты программного обеспечения» / Юго-Зап. гос. ун-т; сост. А.Л. Марухленко. Курск, 2017. - 20с.

Рассматриваются способы отладки программ и обработки ошибок. Указывается порядок выполнения лабораторной работы и содержание отчета.

Методические указания соответствуют требованиям программы, утвержденной учебно-методическим объединением по образованию в области информационной безопасности (УМО ИБ).

Предназначены для студентов специальности 10.03.01.

Текст печатается в авторской редакции

Подписано в печать 01.11.2017. Формат 60x84 1/16.
Усл.печ. л. 1,2. Уч.-изд.л. 1,1. Тираж 30 экз. Заказ _____. Бесплатно.
Юго-Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

ОГЛАВЛЕНИЕ

1. ЦЕЛЬ РАБОТЫ

Целью работы является изучение типов ошибок и возможностей отладки программ.

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1.1 Отладка программы

Успешное завершение процесса компиляции не означает, что в программе нет ошибок. Убедиться, что программа работает правильно можно только в процессе проверки ее работоспособности, который называется *тестирование*.
Обычно программа редко сразу начинает работать так, как надо, или работает правильно только на некотором ограниченном наборе исходных данных. Это свидетельствует о том, что в программе есть алгоритмические ошибки. Процесс поиска и устранение ошибок называется *отладкой*.

1.2 Типы ошибок

При отладке и выполнении программы могут возникать четыре типа ошибок:

Синтаксические. Ошибки, связанные с неправильным синтаксисом оператора (например, If без Then). Ошибки в структуре программы. Ошибки такого типа появляются в результате некорректного написания многострочных операторов (например, For без Next). По сути это синтаксические ошибки. Но Visual Basic обрабатывает ошибки этого типа несколько иначе;

Ошибки, возникающие во время выполнения программы. Они возникают после запуска программы при возникновении ситуации, когда выполнять программу далее невозможно (не обнаруженная при компиляции синтаксическая ошибка, деление на ноль; обращение к гибкому диску, когда устройство не готово).

Логические ошибки. Эти ошибки появляются следствием невнимательности программиста или его заблуждений при разработке алгоритма или при кодировании алгоритма. В итоге программа работает не правильно. Результаты ее работы не соответствуют тестам.

1.3 Синтаксические ошибки

Причиной возникновения синтаксической ошибки могут быть неправильно написанные ключевые слова, ошибки применения разделителей или недопустимые комбинации операторов. Visual Basic распознаёт синтаксические ошибки сразу же после того, как курсор покидает эту

логическую строку. Логическая строка может состоять из нескольких физических строк, разделённых символом подчёркивания:

При обнаружении ошибки Visual Basic выдаёт сообщение с подробным пояснением ошибки. Такие сообщения достаточно информативны и позволяют легко определить причину возникновения ошибки и устранить её.

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ

Задание 1. Проверка синтаксиса

1. Запустите среду VB.
2. Проверку синтаксиса можно включить или отключить с помощью опции *Auto Syntax Check* вкладки *Editor*диалогового окна *Options* меню *Tools.*

3. Зайдите в окно кода формы и напишите такой код: If X = 5. Нажмите Enter – сразу появляется сообщение об ошибке.

4. Теперь отключите проверку синтаксиса (уберите галочку напротив *Auto Syntax Check*) и повторите попытку.
5. Как вы заметили, сообщения не выдалось, хотя и цвет текста предупреждающе изменился (строка с синтаксической ошибкой выделяется красным цветом).
6. Включите проверку синтаксиса.

Примечание. Отключить проверку синтаксиса имеет смысл только в тех редких случаях, когда строка кода формируется путём копирования готовых фрагментов из других мест программы. В этом случае при перемещении курсора в окне кода постоянно появляются раздражающие сообщения об ошибках, причина которых и так известна разработчику. В большинстве случаев отключать проверку синтаксиса не следует.

Задание 2. Контекстная подсказка – QuickInfo

В Visual Basic встроены средства, которые позволяют обнаружить синтаксическую ошибку, но и избежать её при написании кода. Это, в частности, механизм контекстной подсказки или QuickInfo;

1. Напишите в окне кода текст – MsgBox и нажмите пробел (рисунок 1).

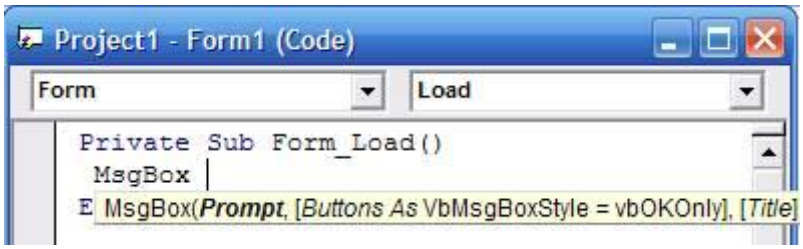


Рисунок 1 – Механизм контекстной подсказки QuickInfo

2. У вас высветилось окно QuickInfo, в котором автоматически отображается полный синтаксис вводимого оператора.

При формировании вложенных вызовов функций вводимый аргумент для удобства всегда выделяется жирным шрифтом (на рисунке 1 показан момент редактирования аргумента *Promt*);

4. Если окно QuickInfo занимает много места и мешает при работе, режим отображения контекстной подсказки можно отключить с помощью опции *Auto Quick Info* вкладки *Editor* диалогового окна *Options* меню *Tools*.

5. Задание 3. Автоматическое отображение списка элементов

6. Для уменьшения количества ошибок при написании имён, свойств и методов объектов, Visual Basic автоматически отображает список доступных элементов. Содержимое списка зависит от типа объекта. Возьмём для примера надпись (*Label*).

1. Поместите на форму надпись и откройте окно кода для формы;
2. Напишите в окне кода следующий текст: Label1.

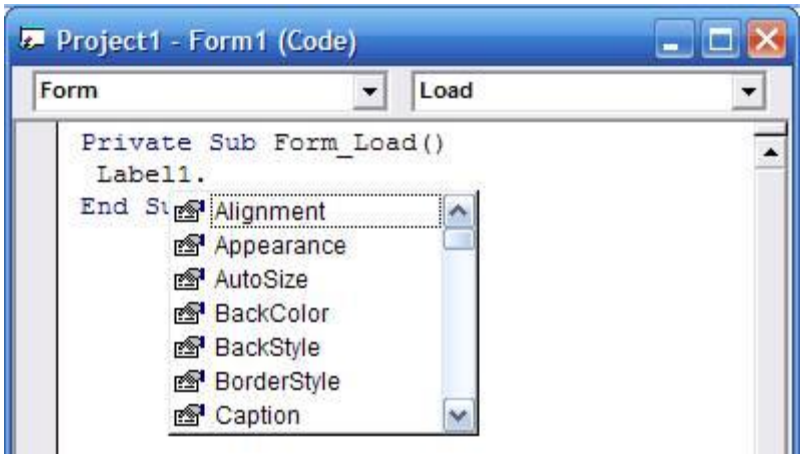


Рисунок 2 – Список свойств и методов

ПОЯСНЕНИЯ

1. Вы можете выбрать элемент из списка или ввести его с клавиатуры. В процессе ввода указатель списка автоматически перемещается к нужному элементу. Тип элемента списка (свойство или метод) указывает пиктограмма рядом с именем.

2. Выбрать нужный элемент в списке можно также с помощью клавиш управления курсором. Нажатием клавиши Tab выделенный элемент вводится в текущую строку, причём текстовый курсор остаётся в этой строке. Для ввода выбранного элемента и перехода на следующую строку следует нажать клавишу Enter.

3. Выберите элемент списка любым предложенным способом.

4. Автоматическое отображение элементов можно отменить. Для этого отключите опцию *Auto List Members* вкладки *Editor* диалогового окна *Options* меню *Tools.*

5. Этот же список можно вызвать. Для этого нажмите комбинацию клавиш Ctrl+J (а список констант открывается комбинацией клавиш Ctrl+Shift+J).

ПРИМЕЧАНИЕ

Аналогично автоматическому списку элементов действует и функция дополнения слова. Если в окне кода введено несколько начальных символов свойства (метода), которых достаточно для их однозначной идентификации, Visual Basic может дополнить недостающие символы. Для этого следует нажать комбинацию клавиш Ctrl+Пробел.

1. Напечатайте в вашем коде Lab и нажмите комбинацию клавиш Ctrl+Пробел. Как вы убедились, Visual basic автоматически дополнил недостающие символы.

2. При вводе кода Visual Basic автоматически устанавливает расстояние между отдельными словами. Напечатайте в окне кода следующий текст:

```
Label1.Caption="Справка"
```

и нажмите Enter. У вас автоматически установились пробелы:

```
Label1.Caption = "Справка"
```

Осторожно! При написании программы не стоит полагаться на то, что Visual Basic сам правильно расставит все пробелы. Например, могут возникнуть сложности при использовании символа коммерческого И, или амперсанда (&). Он может применяться как соединитель строк (в таком случае он отделяется пробелами) или же как идентификатор переменных типа *Long* (используется без пробелов).

3. Напишите в окне кода следующий программный текст:

Label1.Caption = Number& & "Штук; Номер: "

4. Обратите внимание, что в данном примере символ амперсанда (&) выполняет две различные функции. Сначала он служит идентификатором переменной Number типа Long, затем выступает как оператор соединения.

Ошибки в структуре программы

Ошибки в структуре программы – это синтаксические ошибки в многострочных операторах цикла и ветвления. Такие ошибки образуют отдельную группу ошибок, т. к. не распознаются Visual Basic при вводе. Однако при компиляции программы распознавание ошибок такого типа не представляет большой проблемы. В этом случае Visual Basic распознаёт такой незавершённый многострочный оператор, выдаёт сообщение об ошибке и выделяет ошибочный оператор (рис. 7.3.)

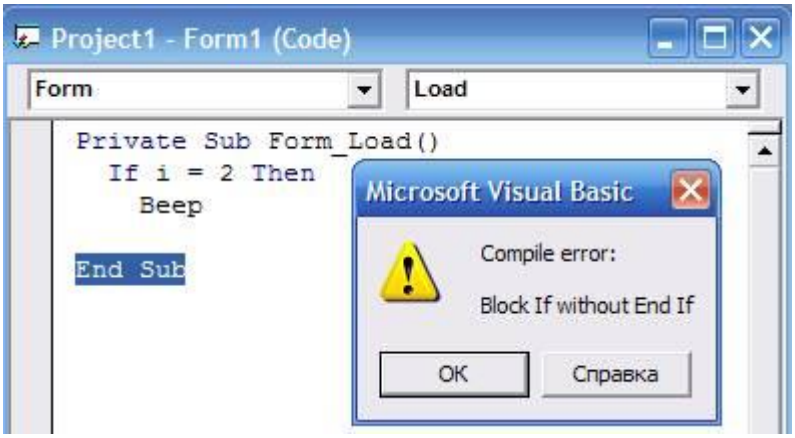


Рисунок 3 – Сообщение об ошибке в структуре программы

Начиная с версии Visual Basic 4.0, приложение не компилируется полностью, если его запускают из среды разработки нажатием клавиши F5 или щелчком на кнопке *Run* панели инструментов. В этом случае ошибки в структуре программы на этапе выполнения выявляются только при обращении к процедуре, содержащей ошибочную структуру. Если же запуск программы осуществляется с помощью команды *Start With Full Compile*меню *Run* или нажатием Ctrl+F5, то все ошибки в структуре программы обнаруживаются сразу при компиляции проекта.

Проверка на отсутствие синтаксических ошибок и ошибок в структуре программы осуществляется и при создании выполняемого файла (*Make*.EXE* меню *File*).

Отладка программы

Это поиск и исправление ошибок в программе. Даже внимательный визуальный анализ программы не всегда позволяет обнаружить допущенную ошибку. В помощь программисту в VB предусмотрено несколько средств, облегчающих поиск ошибок.

О синтаксических ошибках и ошибках выполнения сообщает система, останавливая дальнейшее выполнение программы. Их невозможно не заметить. Логические ошибки коварны. Программа работает, выдает результаты. Но программист обязан заметить, что программа дает неверные результаты. Для этого программист испытывает программу на тестах. Однако, обнаружив дефект в работе программы, следует найти причину этого дефекта. Для этого удобно воспользоваться предоставляемой средой VB возможностью остановить выполнение программы перед выполнением заранее выбранной инструкции и затем продолжить ее дальнейшее выполнение по шагам (по одному оператору) с остановкой после каждого шага и контролем полученных значений переменных.

Интегрированная среда разработки VB предоставляет программисту мощное средство поиска и устранения ошибок в программе – отладчик. Отладчик позволяет выполнять *трассировку* программы, наблюдать значения переменных, контролировать выводимые программой данные. *Трассировка* – это процесс выполнения программы по шагам (step-by-step), инструкция за инструкцией.

Для отладки приложения в пошаговом режиме нужно открыть панель инструментов отладки *Debug* (Отладка), которую можно отобразить, выбрав команду меню *View* (Вид), затем *Toolbars* (Панели инструментов) и щелкнуть на строке *Debug*. На рисунке 4 показана панель инструментов *Debug* (Отладка).

Можно рекомендовать следующую последовательность действий при поиске логических ошибок.

1. Откройте окно *Code* с программным кодом.
2. Откройте панель инструментов отладки.

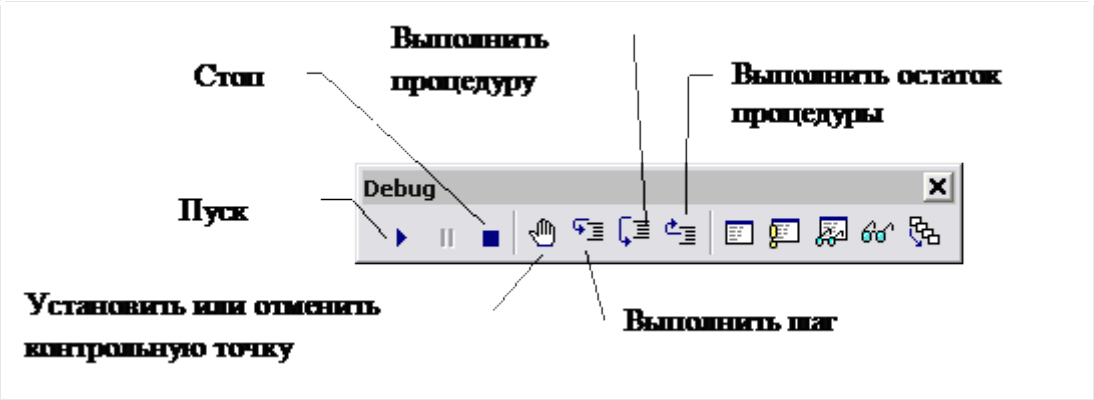


Рисунок 4 – Панель инструментов отладки

3. В подозрительном месте программы, где Вы предполагаете существование логической ошибки, выберите строку программы, перед выполнением которой требуется остановить программу.

4. Установите перед этой строкой контрольную точку. Это можно сделать, например, следующим способом. Щелкните в окне ***Code*** слева от выбранной строки на серой вертикальной полосе. Строка будет выделена коричневым цветом. Для отмены контрольной точки достаточно повторить щелчок.

5. Запустите проект. Выполнение программы будет остановлено перед выполнением строки программы с контрольной точкой

6. После остановки программы можно продолжать ее выполнение по шагам (по одной строке программы). Для выполнения одного шага нужно щелкнуть на кнопке панели инструментов «Выполнить шаг». Если же в следующей строке находится обращение к процедуре или к функции, то они могут быть выполнены за один шаг, если воспользоваться кнопками «Выполнить процедуру» или «Выполнить остаток процедуры».

7. После выполнения каждого шага можно наблюдать значение любой переменной. Для этого достаточно в окне ***Code*** переместить курсор мыши на имя переменной. Появится окно, в котором будет отображено значение переменной, имя которой находится в фокусе мыши.

8. Продолжая так выполнение программы по шагам, Вы, в конце концов, обнаружите то место, где программа делает не то, что Вы ожидали. После этого следует внести необходимые исправления в программу и продолжить отладку до устранения всех ошибок.

2. Упражнения

Упражнение 1

Создадим программу для решения квадратных уравнений и рассмотрим основные способы ее отладки.

Реализация проекта

1. Создайте новый проект.

2. Поместите на форму семь надписей, три текстовых поля и кнопку (рисунок 5).

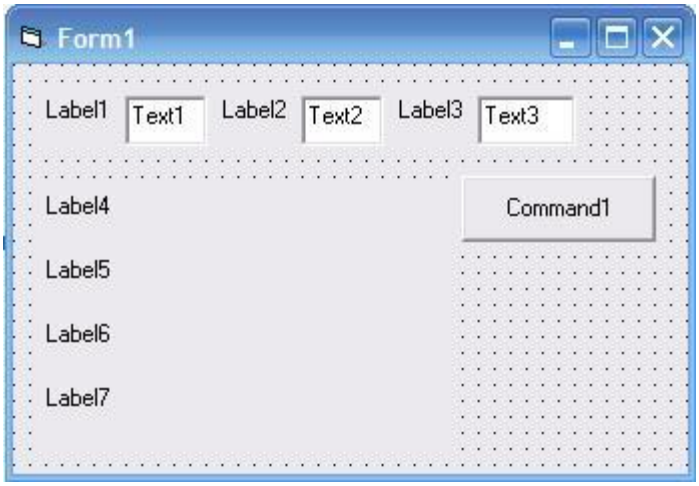


Рисунок 5 – Форма

3. Установите следующие значения свойств объектов:

Класс объектов	**Обозначение объекта по умолчанию**	**Свойство**	**Значение**
Form	Form1	Caption	Решение квадратного уравнения
Label	Label1	Caption	A:
Label2	Caption	B:	
Label3	Caption	C:	
Label4	Caption	Результаты решения:	

Label5	Name	LblD	
Caption			
Label6	Name	LblX1	
Caption			
Label7	Name	LblX2	
Caption			
CommandButton	Command1	Name	CmdCalculate
Caption	Решить		
TextBox	Text1	Name	TxtA
Text			
Text2	Name	TxtB	
Text			

Text3	Name	TxtC
Text		

Ваша форма примет вид (рисунок 6):

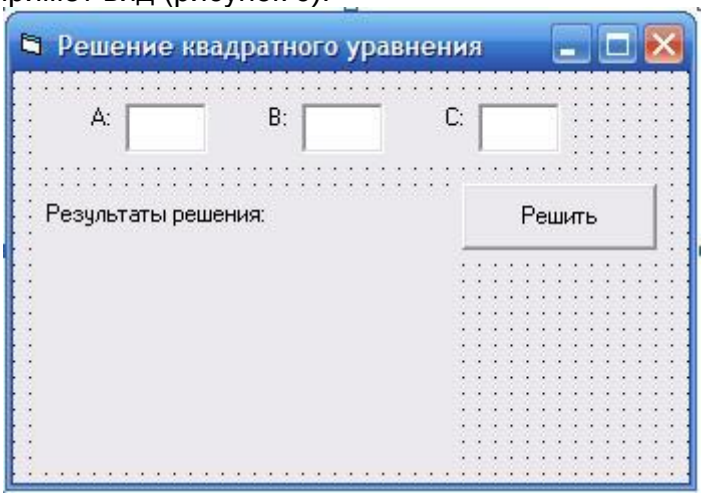


Рисунок 6 – Результат

4. В окне редактора кода введите следующий код:

```
Option Explicit
Private Sub CmdCalculate_Click()
Dim A As Single, B As Single, C As Single
Dim D As Single, x1 As Single, x2 As Single
A = TxtA. Text
B = TxtB. Text
C = TxtC. Text
D = B ^ 2 - 4 * A * C
LblD. Caption = "Дискриминант: " & D
If D > 0 Then
x1 = (B + Sqr(D)) / (2 * A)

x2 = (B - Sqr(D)) / (2 * A)
LblX1.Caption = "Корень №1: " & x1
LblX2.Caption = "Корень №2: " & x2
ElseIf D = 0 Then
x1 = B / (2 * A)
x2 = x1
LblX1.Caption = "Корень №1: " & x1
```

```
LblX2.Caption = "Корень №2 = Корню №1"
Else
LblX1.Caption = "Корней нет!"
LblX2.Caption = ""
MsgBox "Дискриминант меньше нуля! Корней нет!", vbCritical
End If
End Sub
```

5. Сохраните проект.

6. Запустите программу. Введите следующие значения в текстовые поля: $a = 3$, $b = -6$, $c = 2$. Нажмите на кнопку "Решить!". Если вы всё делали правильно, то должны увидеть следующую картину (рисунок 7):

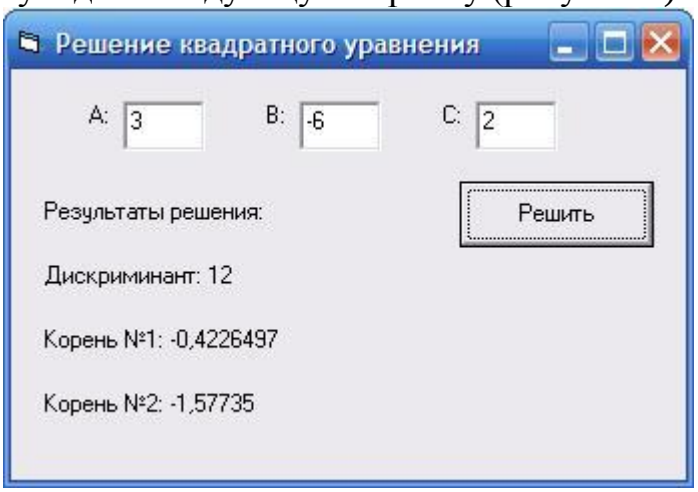


Рисунок 7 – Результат 2

7. Можете поэкспериментировать, вводя разные значения коэффициентов.

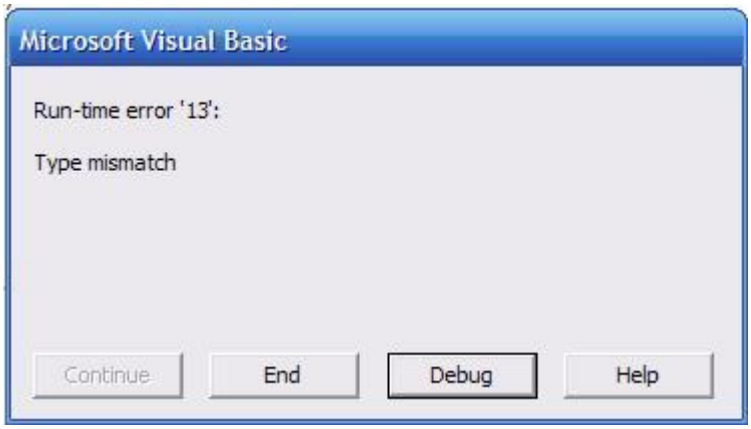
Но, обратите внимание! Что произойдёт, если мы, не введя значения в поля, нажмём на кнопку? Что тогда присвоиться нашим переменным? Или, что будет, если мы введём нули в качестве коэффициентов? В обоих случаях Visual Basic сгенерирует ошибку. Почему?

8. Посмотрим причину возникновения ошибок. Запустите программу. Ничего не вводя в поля, нажмите на кнопку. Visual Basic выдаст окно, в котором скажет: "Type mismatch", т. е. ошибка в типах. В окне доступны 3 кнопки:

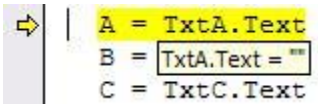
End – завершить приложение

· Debug – показать место возникновения ошибки, чтобы мы смогли от неё избавиться

· Help – вызвать справку о возникшей ошибке.



9. Нажмите Debug. Visual Basic покажет вам причину возникновения ошибки:



Жёлтым цветом выделена строка – причина ошибки. Если навести курсор мыши на имя переменной, то всплывёт подсказка, в которой Visual Basic сообщит нам её значение. Такая возможность доступна только в режиме Debug. Текущий режим можно узнать из заголовка окна Visual Basic.

Например:

в режиме проектировки интерфейса это строка:
Имя_Проекта - Microsoft Visual Basic [design]
при запущенном приложении:
Имя_Проекта - Microsoft Visual Basic [run]
в режиме Debug:
Имя_Проекта - Microsoft Visual Basic [break]

Наведите мышкой на TxtA. Text. VB сообщит о том, что значение текстового поля TxtA. Text = "" (пустым кавычкам, или по другому – пустая строка). Здесь можно догадаться, почему происходит ошибка несовпадения типов. Ведь переменная A должна хранить вещественное число, а здесь мы пытаемся присвоить ей строковое значение – "". Если бы A имела тип String, или в TxtA. Text была бы строка – число ("123") – ошибка бы не возникла. А здесь Visual Basic просто не может сам конвертировать строку "" в число типа Single. Давайте изменим код:

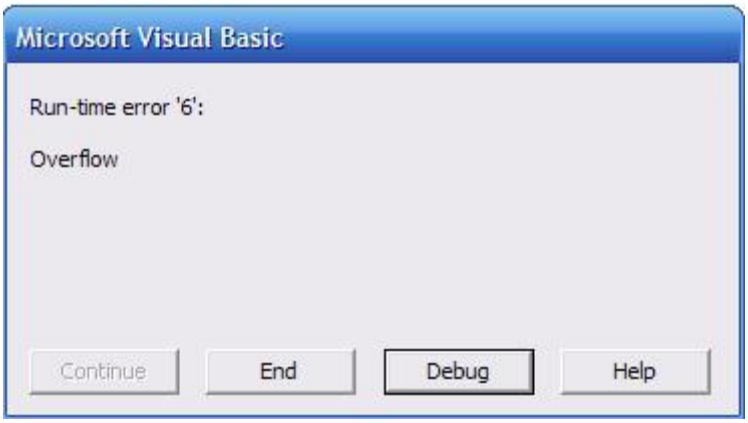
A = TxtA. Text
B = TxtB. Text
C = TxtC. Text

на код

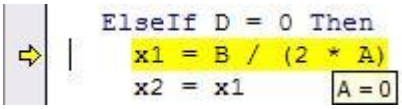
```
A = Val(TxtA. Text)
B = Val(TxtB. Text)
C = Val(TxtC. Text)
```

Теперь ошибок несовпадения типов быть не может, т. к. функция Val при любом параметре возвратит число.

10. Проанализируем полученный результат. Что изменилось после того, как мы вставили функцию Val? Теперь мы знаем, что переменным A, B и C в любом случае присвоится число. Но вот какое число? Ведь если мы введём в поля нули, то и переменные будут содержать нули! Также, если в поля мы введём буквы, то переменные также будут содержать нули. Значит нужно сделать проверку на содержание в переменных нулей. Запустите программу и введите в поля нули (или буквы) и нажмите кнопку. Visual Basic выдаст окно:



Нажимаем Debug и вот что видим:



Причина ошибка заключается в невозможности деления на 0, а A у нас как раз и равен 0. К тому при нулевых коэффициентах квадратное уравнение решается гораздо проще (например, если c =0, то x вынесем за скобку, ну а дальше всё просто). Избавимся от этого недоразумения. Для этого, вставим после присвоения значения свойства Text переменным ещё одну проверку – проверку на содержание нулей в переменных:

```
If A = 0 Or B = 0 Or C = 0 Then
 MsgBox "Нули в качестве коэффициентов не допускаются!", vbCritical
 Exit Sub
```

```
End If
```

Здесь мы проверяем переменные на содержание в них нулей. В принципе, можно было бы проверить не переменные, а сами текстовые поля (If Val(txtParamA. Text)=0 Then...). Это уже дело вкуса. Всё равно, результат одинаков.

Можно запускать программу и решать уравнения.
Итоговый код программы:

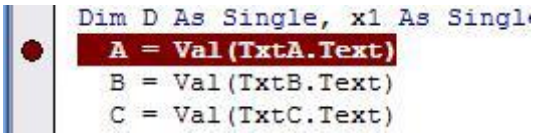
```
Option Explicit
Private Sub CmdCalculate_Click()
Dim A As Single, B As Single, C As Single
Dim D As Single, x1 As Single, x2 As Single
A = Val(TxtA. Text)
B = Val(TxtB. Text)
C = Val(TxtC. Text)
If A = 0 Or B = 0 Or C = 0 Then
  MsgBox "Нули в качестве коэффициентов не допускаются!", vbCritical
  Exit Sub
End If
D = B ^ 2 - 4 * A * C
LblD. Caption = "Дискриминант: " & D
If D > 0 Then
x1 = (B + Sqr(D)) / (2 * A)
x2 = (B - Sqr(D)) / (2 * A)
LblX1.Caption = "Корень №1: " & x1
LblX2.Caption = "Корень №2: " & x2
ElseIf D = 0 Then
x1 = B / (2 * A)
x2 = x1
LblX1.Caption = "Корень №1: " & x1
LblX2.Caption = "Корень №2 = Корню №1"
Else
LblX1.Caption = "Корней нет!"
LblX2.Caption = ""
MsgBox "Дискриминант меньше нуля! Корней нет!", vbCritical
End If
End Sub
```

11. Использование пошаговой трассировки

Пошаговая трассировка – это метод отладки приложения, при котором можно выполнять код по одной команде и следить за ходом её выполнения.

Это очень полезный метод! Таким методом можно находить те ошибки, которые не может найти Visual Basic.

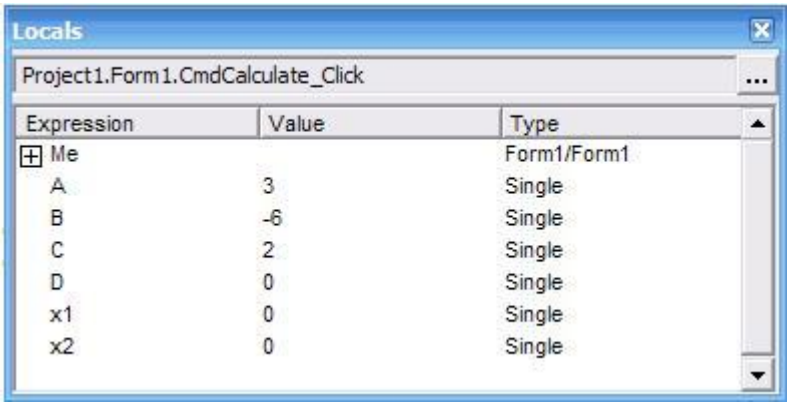
Как использовать такую трассировку? Необходимо в нужном месте программы поставить *точку останова*(брикпоинт (Breakpoint)). Дойдя до такого места, Visual Basic приостановит выполнение программы и переведёт вас в режим Debug, где вы сможете выполнять код покомандно. Давайте поставим брикпоинт в нашей программе, и проследим ход её выполнения. Чтобы поставить точку останова нужно кликнуть на вертикали слева от кода:



А можно и по другому. Нажмите правой кнопкой мыши на той строчке кода, где вы хотите поставить брикпоинт и в меню *Debug* выбрать *Toggle Breakpoint*. Замечание: такую точку нельзя ставить на строчке с объявлением переменной.

Поставьте точку как показано на рисунке и запустите программу. Введите следующие данные в качестве параметров: $a = 3$, $b = -6$, $c = 2$. Нажмите на кнопку "Решить!". Visual Basic остановит выполнение программы точно на той строчке, на которой стоит брикпоинт. Теперь VB в режиме Debug. Вы можете просматривать значения переменных, наводя на них курсор мыши и выполнять программу по шагам. Чтобы выполнить одну команду, нужно нажать F8 или Shift+F8. Отличие этих двух команд заключается в том, что при нажатии F8 на строчке с вызовом процедуры Visual Basic войдёт в эту процедуру, и вы сможете продолжить пошаговое выполнение в этой процедуре до строчки End Sub, где VB вернёт вас на то место, где произошёл вызов. Если же нажать Shift+F8, то Visual Basic выполнит процедуру, но не будет заходить в неё для пошаговой трассировки.

Теперь потрассируйте программу. Проследите за тем, как изменяются значения переменных при присвоении им значения. Теперь откройте окно *Locals Window* меню *View*. В этом окне отображаются значения всех локальных переменных:



Здесь изображены все наши локальные переменные. Их 6 штук. "А вот что такое Me?" - спросите вы. "Me" – это тоже зарезервированное слово Visual Basic. Оно указывает на текущую форму. В своей программе вы можете обращаться к своей форме через Me. Например, если написать

Me. Hide

то с экрана скроется та форма, в коде которой написан этот код.

Также можно заметить значок "+" слева от "Me". При нажатии на него произойдёт раскрытие списка со всеми дочерними объектами формы. Там же будут находиться все элементы управления, помещённые на форму, а также все глобальные переменные уровня этой формы.

Заметьте, что окно *Locals* – плавающее, и при двойном щелчке по заголовку окна оно "пристыкуется" в среду VB. При очередном двойном щелчке окно "отстыкуется".

Дотрассируйте (т. е. пройдите пошагово) программу до конца. Закройте её.

4. ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

1. Изучить теоретические сведение, приведенные в данной работе.
2. Обнаружить синтаксическую ошибку в коде при помощи проверки синтаксиса.
3. Обнаружить синтаксическую ошибку в коде при помощи контекстной подсказки QuickInfo.
4. Осуществить отладку программы.
5. Выполнить упражнения по отладке программы.

5. ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

1. Дайте определение тестированию.
2. Дайте определение отладки.
3. Назовите типы ошибок и дайте им определения.

6. БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Слепцова, Л.Д. Программирование на VBA : Самоучитель / Л.Д. Слепцова. – СПб. : Компьютерное издательство «Диалектика», 2004. – 384 с.

2. Эйткен, П. Разработка приложений на VBA в среде Office XP / П. Эйткен; пер. с англ. – М. : Изд. дом «Вильямс», 2003. – 496 с.

3. Штайнер, Г. VBA 6.3 / Г. Штайнер. – М. : Лаборатория Базовых знаний : Справочник, 2002. – 784 с.

4. Электронный учебник по VBA. Режим доступа: http://www/mini-soft.ru/soft/vba.