Visual Basic – крепкий орешек!

Краткий курс
по изучению языка программирования
Visual Basic

Содержание

Вводный курс в Visual Basic

Приложения

Вводный курс в Visual Basic

Введение.

Итак, вы решили изучить язык программирования высокого уровня - Visual Basic. Я попытаюсь помочь вам в этом. Наберитесь немного терпения, желания и вперёд, в просторы VB! Ведь Visual Basic - это Крепкий Орешек!
Данный курс предназначен для тех, кто

1. никогда не программировал, но хочет научиться
2. программировал на другом языке (Turbo Pascal, QBASIC)
3. программировал на другом языке высокого уровня (C++, Delphi)

От вас требуется только небольшой опыт работы с Windows. И всё.
Этот курс конечно не претендует на полноценный учебник по Visual Basic. Он является введением в VB. Я попыталась разобрать основные моменты и приёмы программирования на Visual Basic, зная которые, вы без особого труда сможете пополнять ваши знания.
То, что не вошло в данный материал, вы сможете прочитать на сайте http://vb.hut.ru. Удачи! И Вперёд!

Урок № 1. Что такое Visual Basic?

Перед начинающими программистами всегда встаёт один и тот же вопрос, а именно, какой язык программирования выбрать? На чём программировать? Могу сказать, что лучше начинать с лёгкого и в то же время мощного языка - Visual Basic. Изучив приёмы программирования на VB, вы сможете без особых усилий изучить другие языки, такие как Pascal, C++ и др.
Слово "БЕЙСИК" (BASIC) - "базовый, основной" - образовано из начальных букв английского выражения "Универсальный язык символического кодирования для начинающих". Это "для начинающих" долго вызывало пренебрежение программистов, причём подобное пренебрежение не исчезло до сих пор, несмотря на наличие префессиональных изданий VB.
Идём дальше. Что же может Visual Basic?

Урок № 2. Что может Visual Basic?

В принципе, возможности Visual Basic ничем не ограничены. Вы можете расширять возможности VB посредством использования дополнительных функций.
На VB можно написать любую программу, от обслуживающих рутинные операции ввода данных, до сложных информационных и коммуникационных систем. В США 60% программных продуктов написаны на VB. Так что просторы VB очень велики!
Теперь можно приступать к установке и настройке VB!

Урок № 3. Установка и настройка VB (рекомендации)

Установка Visual Basic не отличается особой сложностью, всё стандартно. В процессе инсталляции вас попросят указать компоненты, которые будут установлены на ваш компьютер. Если на жёстком диске есть место, то лучше выбрать все компоненты, чтобы потом их не пришлось добавлять. Здесь следует отметить то, что версия VB 5.0 поставляется с файлами помощи (примерно 15Мб), а помощь для VB 6.0 идёт в комплекте с MSDN, который поставляется на трёх дисках. Поэтому если у вас шестая версия, записанная на одном диске, то будьте уверены, помощи там не будет. Но вполне можно взять помощь от пятой версии и использовать её для VB 6.0. После установки VB перезагрузите компьютер.
Перед работой с VB его необходимо настроить. Для этого запустите VB (Пуск->Программы->Microsoft Visual Basic 6.0->Visual Basic 6). Зайдите в меню Инструменты->Параметры. Поставьте галочку "Require Variable Declaration". Это избавит вас от лишних ошибок при автоматическом определении переменных. Далее на вкладке Editor Format, в списке Font укажите Courier New Cyr. Если этого не сделать, то VB не будет корректно отображать кириллицу. Также рекомендую установить цвет зарезервированных слов в ярко-синий. Для этого выберите в списке Code Colors Keyword Text и в поле Foreground укажите ярко-синий цвет (седьмой снизу). Вот и всё! Visual Basic готов к работе!

Урок № 4. Для тех, кто никогда не ...

Закон программирования гласит: "Ни одна, даже самая простая программа, не работает сразу после написания". Любую программу необходимо отлаживать (обезжучивать - debug (bug-жук)). На этот счёт

имеется ещё закон: легче написать свою собственную программу, чем разбирать и исправлять чужую. Отладка программы состоит в следующем:

1. Запустить программу.
2. В случае сбоя или неправильной работы найти причину ошибки.
3. Устранить ошибку.
4. Продолжать до тех пор, пока не будут устранены все ошибки.

Урок № 5. Этапы разработки приложения

В Visual Basic, как и во многих других языках, предназначенных для написания приложений под Windows, используется событийно-управляемая модель программирования. Графический интерфейс пользователя имеет снандартные элементы управления, такие, как окна (они же формы), кнопки, списки, поля, для ввода текста и т.п. В любом языке высокого уровня программа строится на основе этих элементов. Итак, разработка приложения на VB состоит из следующих этапов:

1. Продумывания программы (подумать, что программа должна делать, решить перед собой задачи, реализовать их мысленно, продумать структуру данных, и т.д.).
2. Проектирование интерфейса, т.е. помещение на форму нужных управляющих элементов, кнопок, списков и т.п. Этот этап называется составлением скелета программы.
3. Написание программного кода, связывающего помещённые на форму управляющие элементы, т.е. "наращивание плоти на скелет".
4. Отлаживание программы. Этот этап часто занимает больше времени, чем предыдущие.
5. Окончательная компиляция и, если это необходимо, создание дистрибутива (т.е. установочного файла setup.exe).

Урок № 6. Структура проекта VB (обратно)

В Visual Basic любой проект состоит из следующих файлов:

- файл каждой формы (расширение frm), текстовый файл, в котором записан весь код свойств всех помещённых на форму элементов управления и самой формы.
- файл каждой формы, содержащий бинарную информацию (например, картинку в PictureBox) (расширение frx).
- файл проекта, содержащий информацию о проекте (расширение vbp).
- информация о рабочей области проекта (workspace) (расширение vbw).

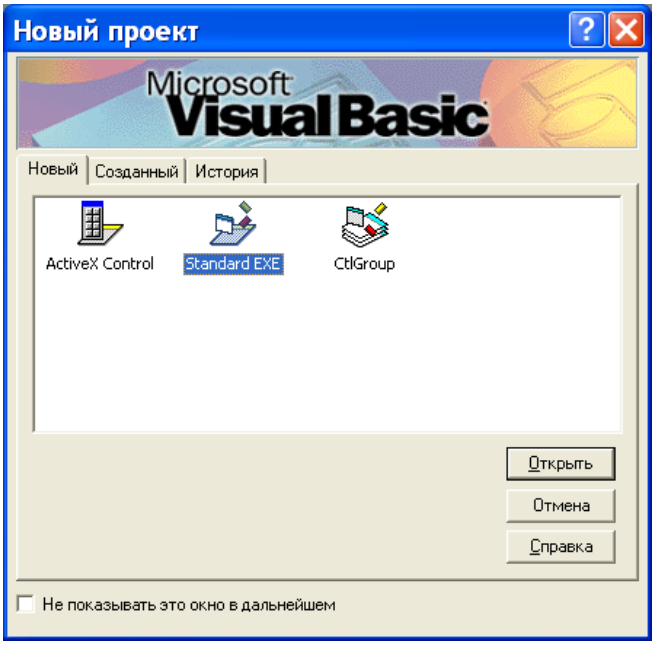
Дополнительные файлы, которые могут быть подключены к проекту:

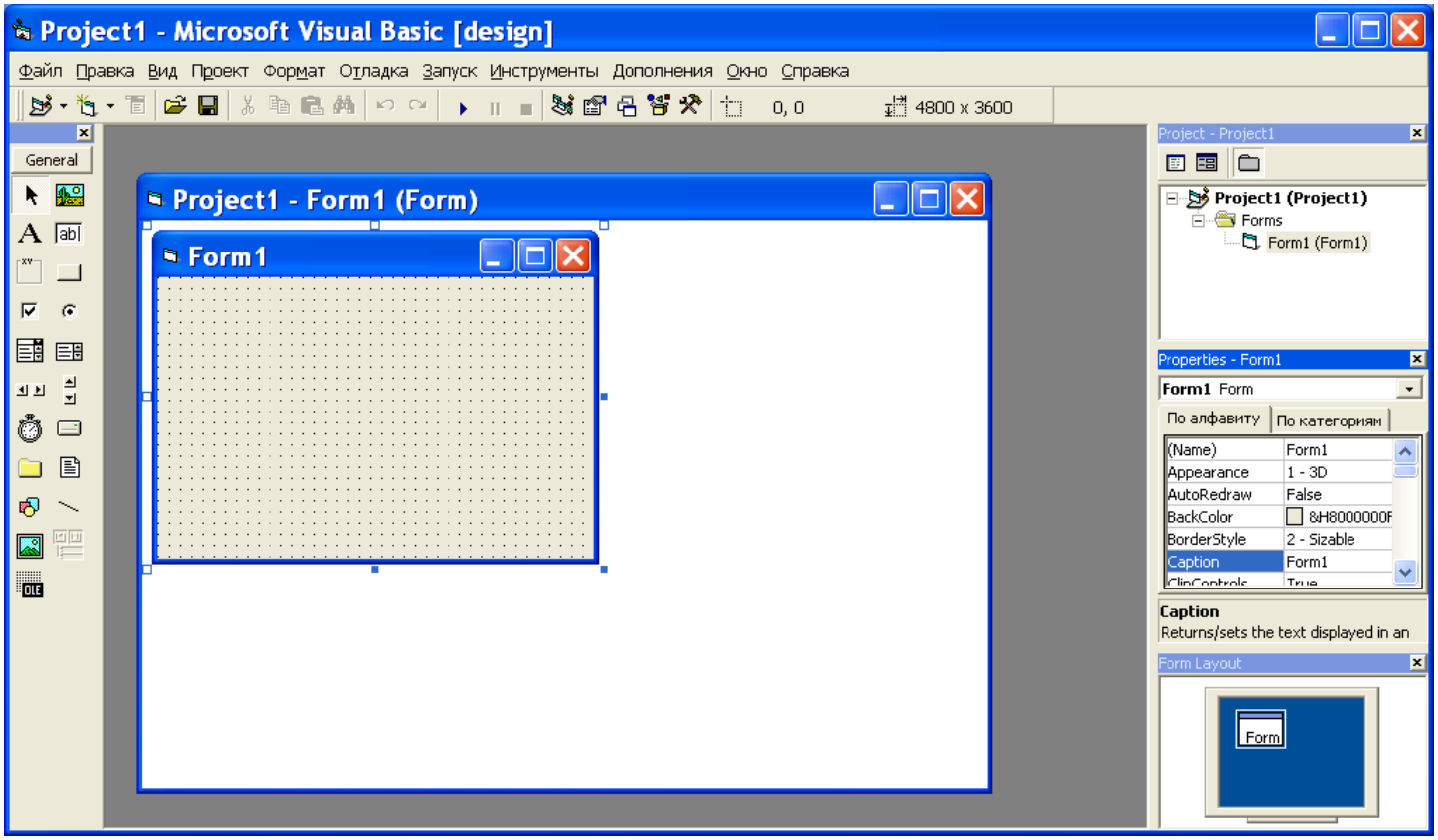
- файл каждого модуля (расширение bas) Это текстовый файл.
- файл каждого модуля классов (расширение cls). Это текстовый файл.
- файл каждого дополнительного элемента управления (расширение ctl) Это тоже текстовый файл.
- файл ресурсов (расширение res)
- другие файлы (ocx, tlb, и т.д...)

Запоминать назначение всех этих файлов не обязательно, достаточно запомнить 2 файла: frm-файл, в котором хранятся код формы и свойства всех помещённых на данную форму элементов управления. И bas-файл - модуль. В нём могут быть объявлены глобальные переменные, константы, функции и т.д.

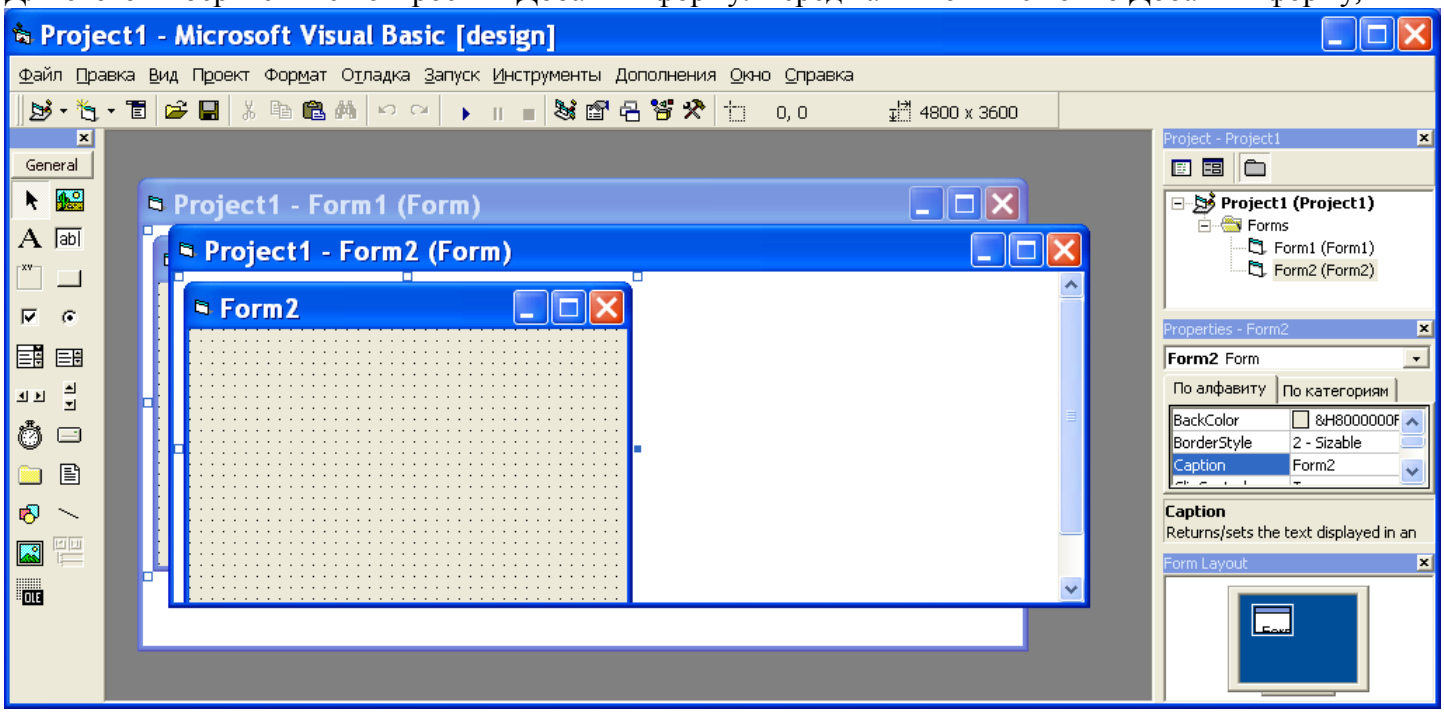
Урок № 7. Среда разработки VB (обратно)

Для того, чтобы понять как проектировать интерфейс, разберём для начала среду разработки Visual Basic. Запустите VB. Перед вами появится окошко, в котором вам попросят указать тип проекта. Укажите тип "Standart EXE" и нажмите ОК. На экране вы увидите следующее окно:



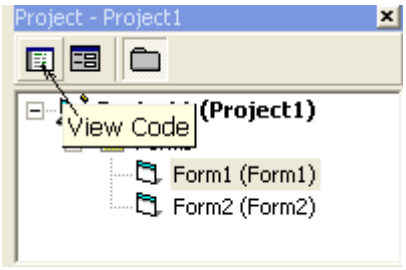


В левой части расположена панель с доступными элементами управления, с которой вы можете перетаскивать нужные элементы на форму. В центре находится форма (окно) вашего приложения. Имя новой формы - Form1. Вверху расположена панель инструментов среды разработки. Справа расположены окна проекта (Project) и свойств текущего объекта (Properties). Здесь необходимо отметить, что все объекты в VB (впрочем, как и в других языках высокого уровня) имеют свойства и методы. Свойства — значения, которые устанавливаются для определения вида и поведения объекта. Методы — программные процедуры, обеспечивающие выполнение объектом некоторых действий.
Теперь закройте программу. Это можно сделать двумя способами - нажать на крестик в правом верхнем углу формы или нажать на кнопку End, имеющей иконку кнопки Stop.
После закрытия программы вы вернётесь в среду разработки VB. Кстати, обратите внимание на окошко Project. В нём показан только один файл - Form1. Давайте добавим ещё одну форму к нашему проекту. Для этого выберите в меню Проект->Добавить форму. Перед вами появится окно Добавить форму,



в котором вам предложат выбрать вид новой формы. Дважды кликните на иконке с надписью Form.

Перед вами появится новая форма. Её имя Form2. Но куда же делась старая, спросите вы? Для того, чтобы увидеть нашу старую форму, нужно дважды щёлкнуть по строчке Form1 в окне Project. Щёлкнув, вы сразу увидите нашу первую форму. Обратите внимание на 2 кнопочки в окне Project. View Object и View Code. Эти кнопочки вам очень пригодятся в дальнейшем. С помощью них вы можете переключаться между двумя режимами:



1. просмотром формы, для проектирования её интерфейса,
2. просмотром кода формы.

Урок № 8. Лёгкость работы с кодом в VB

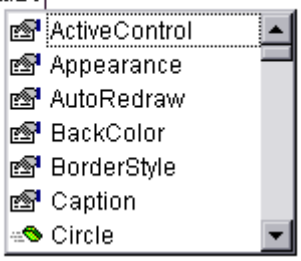
Технология Intellisence сильно облегчит вам жизнь в процессе программирования на VB. Эта технология Microsoft позволит вам избежать ввода большого количества кода и его корректировки. Intellisence выводит небольшое всплывающее окно с полезной информацией о текущем объекте. Такие окна бывают 3-х видов:

1. QuickInfo. Выдаёт информацию о синтаксисе текущего оператора Visual Basic. Где бы вы не ввели имя оператора (функции) и поставили после имени пробел или октрывающую круглую скобку, то Visual Basic незамедлительно покажет информацию о синтаксисе этого оператора. Вот пример:

```
D = CalcDiscremenant (
    CalcDiscremenant(a As Double, b As Double, c As Double) As Double
If D > 0 Then
```

2. List Properties/Methods. Это свойство облегчит вам работу с объектами в Visual Basic. После того, как вы поставите точку после имени какого либо объекта, VB сразу же покажет вам список всех доступных свойств и методов этого объекта:

```
Private Sub Form_Load()
    Form1.
End Sub
```



Свойства имеют иконку:  , а методы -  .

3. Available Constants. Эта функция выводит окно доступных констант. Например, если вы поставите знак равенства после Boolean переменной, то Visual Basic выдаст вам окно, где вы сможете выбрать из двух значений (True/False) нужное. Вам даже не придётся ничего набирать на клавиатуре!

4. Также, Если нажать Ctrl+J, то VB выдаст список всех определенных в программе свойств, методов, констант, типов и т.д, включая встроенные в сам Visual Basic.

Отступы

Про отступы следует поговорить отдельно. Отступы ОЧЕНЬ выжны при программировании. НИкогда НЕ забывайте про них! Они помогут при просмотре вашего кода. Особенно они полезны в сложных ветвлениях и циклах.

Visual Basic предоставляет возможность сделать отступ сразу для участка кода. Для этого необходимо выделить этот участок (несколько строк) и нажать Tab. Все выделенные строки сдвинуться вправо. Если вам понадобиться сдвинуть код влево, нажимайте Shift+Tab. Всегда помните про отступы!

Урок № 9. Из чего состоит код?

Итак, мы научились проектировать интерфейс программы. Но для полноценной программы этого не достаточно. Нужно написать код программы, который будет манипулировать элементами управления, и производить какие-то вычисления. Это самый сложный этап.

Во всех языках высокого уровня программный код состоит из:

- Переменных
- Операторов
- Функций
- Выражений
- Управляющих структур
- Классов и объектов

Опишем каждый тип подробнее:

Переменные.

В Visual Basic переменые хранят информацию (значения). При их использовании Visual Basic резервирует область в памяти компьютера для хранения данной информации. Каждая переменная имеет своё имя. Оно может достигать 255 символов в длину, начинается всегда с буквы латинского алфавита, за которой могут следовать другие буквы, цифры и знак подчёркивания.

Каждая переменная имеет определённый тип. Всего в VB 14 типов переменных. Перечислим основные типы переменных VB:
Byte - предназначен для хранения целых чисел от 0 до 255. Если переменной такого типа присвоить значение, выходящее за эти пределы, то Visual Basic сгенерирует ошибку.
Integer - предназначен для хранения целых чисел в диапазоне -32768 до +32767, т.е. размер памяти, выделяемой под такую переменную состовляет 2 байта. (256*256=65536). Символ - "%".
Long - предназначен для хранения целых чисел в диапазоне -2147483648 до +2147483647, т.е. размер памяти, выделяемой под такую переменную состовляет 4 байта. (65536*65536=4294967296). Символ для обозначения - "&".
String - предназначен для хранения строковой (символьной) информации, т.е. попросту говоря - текста. Может хранить до 2 Гб. текста. Символ для обозначения - "$".
Double - предназначен для хранения дробных чисел, с точностью до 16 цифр. Диапазон отрицательных значений от 1.79769313486232E308 до -4.94065645841247E-324. Диапазон положительных значений от 4.94065645841247E-324 до 1.79769313486232E308. Длина числа может достигать 300 знаков. Занимает 8 байта памяти. Вычисления с данными переменными будут приблизительными и менее быстрыми, чем с переменными целого типа. Используется для научных рассчётов. Символ для обозначения - "#".
Currency - Данный тип создан для того, чтобы избежать ошибок при преобразовании чисел из десятичной формы в двоичную и наоборот (Невозможно представить 1/10 как сумму 1/2, 1/4 и т.д). Данный тип может иметь до 4 цифр после запятой, и до 14 перед ней. Внутри данного диапазона вычисления будут точными. Вычисления выполняются так же медленно, как и в случае переменных Single и Double. Данный тип очень подходит для финансовых расчётов. Символ для обозначения - "@".
Date - Этот тип данных позволяет хранить значения времени и даты в промежутке от полуночи 1 января 100 года до полуночи 31 декабря 9999 года. Если переменной присвается только значение даты, то время равняется 00:00.
Если не указывать As Имя_Типа, то переменная будет объявлена как Variant.

Урок № 10. Массивы, записи и перечисления

На предыдущем уроке мы рассмотрели с вами такую важную вещь, как переменные. А переменные - это ведь кусочки памяти, где хранятся данные. Значит, если эффективно использовать переменные - мы эффективно используем память. А если мы эффективно используем память - то памяти для приложения нужно меньше и приложение работает быстрее. Так вот для того, чтобы эти данные использовать с максимальной эффективностью, и в то же время с лёгкостью, были придуманы "массивы" (Arrays), "записи" (Types) и "перечисления" (Enums).

Массивы

Их ещё называют списками. Итак, что же такое массивы? Массив - это набор однотипных переменных, объединенных одним именем и доступных через это имя и порядковый номер переменной в наборе. Количество элементов массива теоретически может быть бесконечным; ограничения накладываются конкретным языком программирования и операционной системой. Элементы массива обладают непрерывной нумерацией определённого диапазона. В программировании массивы используются довольно часто.
В Visual Basic массивы определяются следующим образом:

```
Dim myArray (10) As Long
```

Записи

Запись - это новый, опередяемый программистом тип данных, который состоит из одной и более переменных внутри. Давайте рассмотрим это на примере. Например, необходимо в программе хранить массив студентов. Причём каждый студент имеет свои характеристики: ФИО, Возраст, Ниличие Грамот. Конечно, для хранения таких данных можно использовать, например, массив, имеющий две размерности. Но это не лучший вариант. Лучше всего здесь подходят Записи! Затем из записи можно будет сделать массив! Чтобы определить запись в программе нужно использовать зарезервированное слово Type. Заканчивается запись словами End Type:

```
Private Type Student ' вместо Private могло быть и Public
    FIO As String
    Age As Byte
    HasGramot As Boolean
End Type
```

Перечисления

Перечисление - это список констант. Перед использованием такого списка его необходимо определить в программе. Например, рассмотрим перечисление оценок, получаемых студентами:

```
Enum Ocenka
    Neud = 3
    Horosho = 4
    Otlichno = 5
End Enum
```

Присваивать значения константам внутри Enum не обязательно. Если этого не сделать, то константы будут принимать значения 0,1,2... и т.д.
Теперь можно объявить переменную типа Ocenka:

```
Dim oc1 As Ocenka
```

Урок № 11. Выражения

В любом языке программирования выражения являются основными кирпичиками, из которых строится программа. Согласно самому точному определению, "выражение" - это "что-то, что содержит значение". За примером далеко ходить не нужно, возьмём пример из предыдущего урока:

b = 234

Здесь мы присваиваем переменной b значение 234. Другими словами "234" - это выражение со значением 234. А теперь, к примеру, рассмотрим строчку:

c = b

Здесь переменной c присваивается выражение b. Значение этого выражения - b = 234. Т.е. другими словами b - выражение, со значением 234. Рассмотрим более сложный пример выражения - функцию. Объявим функцию MyFunc, возвращающую байт 234:
А теперь запишем строку:

c = MyFunc()

Как вы уже, наверное, догадались, выражение здесь - MyFunc(), со значением 234. Т.е. после присвоения переменной c выражения MyFunc(), она будет содержать значение 234. А вот ещё пример:

c = 5 + 5 * 2

Здесь выражение это 5 + 5 * 2. Значение этого выражения не трудно посчитать, оно равно 15. Можно было бы написать и так:

c = MyFunc() - 219

Здесь значение выражение такое же, как и в предыдущем случае, но вот сами выражения разные. Это важно понимать.
Также необходимо отметить значение скобок в выражениях. В Visual Basic скобки выполняют ту же функцию, что и в школе, а именно - задают приоритет операции. К примеру, модифицируем выражение 5 + 5 * 2 на:

c = (5 + 5) * 2

Теперь значение этого выражения не 15, а 20!

Урок № 12. Операторы

В Visual Basic операторы бывают следующих типов:

- Арифметические:
  - ^ оператор возведения в степень.
  - * оператор умножения.
  - / оператор деления
  - \ оператор целочисленного деления
  - Mod оператор вычисления остатка от деления
  - + оператор сложения
  - - оператор вычитания
- Сравнения:
  - < меньше
  - > больше
  - <= меньше или равно
  - >= больше или равно
  - = равно
  - <> не равно
- Конкатенации:
  - + оператор конкатенации
  - & оператор конкатенации
- Логические:
  - And оператор логического умножения
  - Eqv оператор логической эквивалентности
  - Imp оператор логической импликации
  - Not оператор логического отрицания
  - Or оператор логического сложения
  - Xor оператор логического исключающего сложения

Урок № 13. Управляющие структуры

Примечание: Здесь бы хотелось заметить, что управляющие структуры мы тоже будем называть операторами. Операторы, рассмотренные на предыдущем уроке, используются в выражениях. А операторы, рассматриваемые на этом уроке, предназначаются для управления вычислением этих выражений. Важно понимать различие между этими операторами.

1.Условный оператор If...End If

Этот оператор необходим для принятия решений, нужно ли выполнять то или иное действие или нет. Другими словами если `Логическое_выражение` истинно, то `Оператор` выполнится. Если ложно, то выполнение не произойдёт.

```
If Логическое_выражение Then Оператор

или сложнее

If Логическое_выражение Then
    Группа_операторов
End If
```

В первом случае оператор может быть только один. Во втором сколько угодно (в том числе и один).
Пример:

```
If (a = b) And (c <> d) Then
    b = d
    a = 20
End If
```

Скобки здесь не обязательны, но они повышают читабельность кода.

2. Условный оператор Select Case...End Select

Конструкция Select Case "принимает решение" на основе анализа значения одного выражения. При этом это выражение указывается в строке Select Case:

```
Select Case Анализируемое_выражение
```

```
Case Значение№1
    Группа операторов
Case Значение№2
    Группа операторов
...
Case Значение№N
    Группа операторов
Case Else
    Группа операторов

End Select
```

Конечно, анализируемое выражение должно возвращать значение типа, совместимого с типом значений в строке Case.

3. Оператор цикла For...Next

Этот цикл используют в том случае, когда заранее известно стартовое и конечное значение счётчика. Синтаксис выглядит следующим образом:

```
For Счётчик_цикла = Старт To Стоп Step Шаг
    Группа операторов
Next [Счётчик_цикла]
```

Роль счётчика цикла может играть только ранее объявленная переменная целочисленного типа. Шаг задаёт приращение счётчика цикла при каждом проходе. Умолчательно значение шага равно 1. После слова Next счётчик можно опустить.

Пример:

В этом примере всем элементам массива iArray присваивается значение 5.

```
Dim c As Integer
Dim iArray(10) As Integer
For c = 0 To 10
    iArray(c) = 5
Next c
```

3. Оператор цикла For Each...Next

Эта специфическая форма цикла For предназначена для выполнения некоторой операции с каждым объектом, входящим в состав некоторой коллекции объектов (такой операцией, например, может быть вызов метода или присваивание значения свойству). Синтаксис оператора:

```
For Each ИмяОбъекта In ИмяКоллекции
    Операции над объектами
Next ИмяОбъекта
```

Пример:

В этом примере показано, как изменить свойство BackColor у всех этикеток (Label), лежащих на форме

```
Dim x As Object
For Each x In Me.Controls
    If TypeName(x) = "Label" Then
        x.BackColor = 0
    End If
Next x
```

Me здесь - текущая форма. Т.е. не обязательно использовать полное имя формы для доступа к её свойствам. Например, для закрытия текущей формы, можно написать Me.Hide. (или Unload Me).

4. Оператор цикла Do While...Loop / Do...Loop While

Эти две разновидности цикла тесно взаимосвязаны, и их часто рассматривают как один из базовых видов цикла. Как уже отмечалось, циклы For применяют в тех случаях, когда количество проходов и диапазон изменения счётчика цикла заранее известны. Циклы While предназначены для ситуаций, когда

количество проходов цикла заранее не известно, но зато известно условие выхода из цикла. Синтаксис цикла While:

```
Do While Условие_выхода
    Группа операторов
Loop

Do
    Группа операторов
Loop While Условие_выхода
```

5. Оператор цикла Do Until...Loop / Do...Loop Until

По своей логике цикл Until подобен циклу While с той лишь разницей, что проходы цикла выполняются до тех пор, пока условие выхода не выполняется.

Пример:

```
Dim n As Integer
n = 100
Do
    n = n - 1
    Debug.Print n
Loop Until n < 11
```

6. Выход из цикла Exit For / Exit Do

С помощью операторов Exit... можно осуществить досрочный выход из цикла вне зависимости от значения, которое имеет в данный момент условие выхода.

Пример:

```
Dim n As Integer
n = 10
Do While n > 1
    n = n - 1
    Debug.Print n
    If n = 5 Then Exit Do ' Если счётчик = 5, то
                          'выходим из цикла
Loop
```

Итак, управляющие структуры - очень важное и далеко не слабое звено в программировании на Visual Basic (да и не только на Visual Basic). Без использования таких структур не получится написать даже самую маленькую программу. Даже если и получится, то программа не будет представлять никакого практического интереса.

Урок № 14. Процедуры и функции

Выражения и операторы - это сырьё для блоков, из которых строится программы, где в роли блоков выступают процедуры и функции.

Процедуры и функции

В Visual Basic, как и во многих других языках программирования, большинство программ создается из блоков - процедур и функций. Весь программный код находится как бы внутри этих процедур. Если возникает необходимость в решении какой-либо задачи в любом месте программы, то вызывается процедура. В Visual Basic нельзя ввести код между процедурами. Код всегда должен находиться внутри процедуры.

Давайте разберёмся с понятиями, и определим, что будет называться процедурой, а что функцией.

Процедуры:

Процедура - это некий блок кода, который будет выполняться всякий раз при вызове этой процедуры. Каждая процедура начинается зарезервированным словом Sub и заканчивается End. Вот общий синтаксис процедуры:

```
[Private | Public | Friend] [Static] Sub name [(arglist)]
    [здесь некий код]
    [Exit Sub]
```

```
    [здесь тоже может быть некий код]
End Sub
```

arglist имеет следующий вид:

```
[Optional] [ByVal | ByRef] [ParamArray] varname[( )]
[As type] [= defaultvalue]
```

Функции:

Функция - это некий блок кода, который будет возвращать значение. Этим, и только этим функции отличаются от процедур. Общий синтаксис функции:

```
[Public | Private | Friend] [Static] Function имяфункции _
[(arglist)] [As type]
    [здесь некий код]
    [имяфункции = выражение]
    [Exit Function]
    [здесь тоже может быть некий код]
    [имяфункции = выражение]
End Function
```

Что значит "будет возвращать значение"? Рассмотрим функцию из урока 8:

```
Public Function MyFunc() As Byte
    MyFunc = 234
End Function

c = MyFunc()
```

Когда мы говорили о выражениях, мы говорили, что MyFunc - это выражение, со значением 234. Т.е. здесь, функция MyFunc возвращает значение 234 (байт). Чтобы задать это значение, необходимо присвоить имени функции выражение. В нашем случае в качестве выражения выступает число 234. Давайте рассмотрим более практичный пример. Напишем функцию для вычисления квадрата числа. У функции будет 1 параметр типа Integer - число для возведения в квадрат. Функция будет возвращать значение квадрата параметра. Тип возвращаемого значения - Long:

```
Public Function Square(number As Long) As Long
    Square = number * number
End Function
```

Вызвать функцию можно так:

```
b = Square (5)
```

А можно так, используя нашу процедуру для вывода сообщения на экран:

```
ShowMessage Square (5)
```

А можно и так:

```
Square 5
```

Урок № 15. Сводим всё вместе

В этом уроке мы попробуем написать нашу первую программу на Visual Basic - программу для решения квадратных уравнений. Может быть, эта программа и не очень полезна в хозяйстве, но она хорошо вас ознакомит с принципами программирования на VB.

Всмомним из урока 5 основные этапы разработки приложение на Visual Basic:

1. Продумывание программы
2. Проектирование интерфейса
3. Написание программного кода
4. Отлаживание программы
5. Окончательная компиляция

Программу будем писать согласно этим пунктам:

1. Продумывание программы.

Что должна делать наша программа? - решать квадратные уравнения. Вспомним, как решаются квадратные уравнения.

a*x*x + b*x + c = 0

Чтобы решить такое уравнение, нужно найти его дискриминант и затем корни. Дискрименант:

D = b*b - 4*a*c

Корни:

если дискриминант > 0, то X1 = (-b + (корень из D)) / 2*a, X2 = (-b - (корень из D)) / 2*a

если дискриминант = 0, то X1 = X2 = -b / 2*a

если дискриминант < 0, то корней не существует.

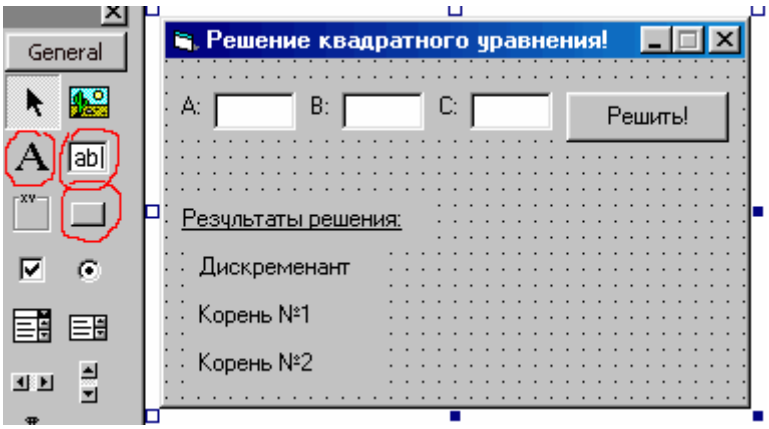
Итак, что вышло:

Входные данные в программу - коэффициенты a,b,c. Данные будем вводить в текстовые поля для ввода (TextBox).

Выходные данные - корни (если они есть) и дискриминант. Данные будем выводить в элемент метку (Label). Запускать решение будем нажатием на кнопку (Command Button).

2. Проектирование интерфейса.

Запустите Visual Basic. Выберите тип Standart EXE. Поместите на форму элементы управления, как показано на рисунке:



Давайте переименуем элементы управления. Для этого измените свойство Name каждого элемента управления согласно таблице:

Назначение элемента управления	Имя (свойство Name)
A: - параметр A	txtParamA
B: - параметр B	txtParamB
C: - параметр C	txtParamC
Кнопка для запуска решения	cmdCalculate
Label, с вычисленным дискриминантом	lblD
Label, с корнем X1	lblX1
Label, с корнем X2	lblX2
Форма, содержащая все эти элементы	frmMain

Остальные элементы переименовывать не обязательно, т.к. мы к ним не будем обращаться в коде программы. Но, если хотите, можете переименовать. Приучайтесь к этому.

3. Написание программного кода.

Теперь самое интересное! Мы будет писать код для нашей программы! Давайте ещё раз продумаем алгоритм работы программы:

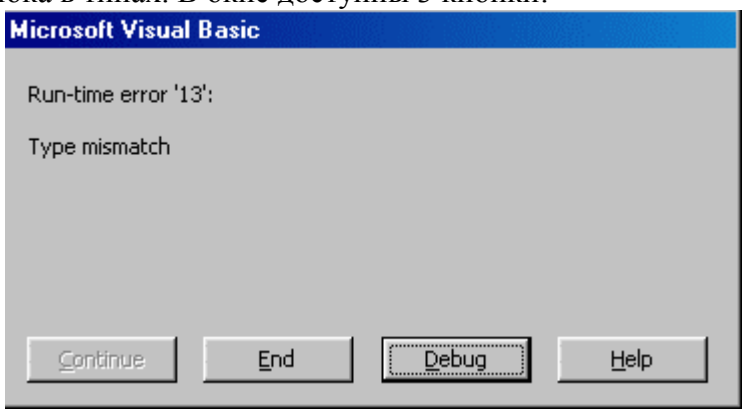
1. Вводим исходные данные в тектовые поля (a,b,c). Код для этого писать не нужно. За нас всё сделает Visual Basic и Windows. В этом то и заключается прелесть графического интерфейса пользователя (GUI). Мы только считаем введенные значения и всё.

2. После нажатия на кнопку, производим вычисление дискриминанта и корней.
3. Выводим полученные значения в метки (Label'и).

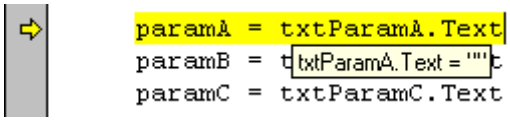
```
Private Sub cmdCalculate_Click()
    ' объявляем переменные
    Dim paramA As Double
    Dim paramB As Double
    Dim paramC As Double

    Dim x1 As Double
    Dim x2 As Double
    Dim D As Double

      'считаем введённые параметры a, b и c
      paramA = txtParamA.Text
      paramB = txtParamB.Text
      paramC = txtParamC.Text
      'рассчитаем дискриминант
      D = (paramB * paramB) - (4 * paramA * paramC)
      'вычислим корни, если они существуют
If D > 0 Then
    x1 = (- paramB + Sqr(D)) / (2 * paramA)
    x2 = (- paramB - Sqr(D)) / (2 * paramA)
    lblD.Caption = "Дискременант: " & D
    lblX1.Caption = "Корень №1: " & x1
    lblX2.Caption = "Корень №2: " & x2
ElseIf D = 0 Then
    x1 = paramB / (2 * paramA)
    x2 = x1
    lblD.Caption = "Дискременант: " & D
    lblX1.Caption = "Корень №1: " & x1
    lblX2.Caption = "Корень №2 = Корню №1"
ElseIf D < 0 Then
    lblD.Caption = "Дискременант: " & D
    lblX1.Caption = "Корней нет!"
    lblX2.Caption = ""
    MsgBox "Дискременант меньше нуля! Корней нет!", vbCritical
End If

End Sub
```

Урок № 16. Отладка программы

Итак, на предыдущем уроке мы нашли баги (от слова Bug - жук) в нашей программе, т.е. недочёты (ошибки). От этих багов нужно избавиться. 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.
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 возможноть 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:

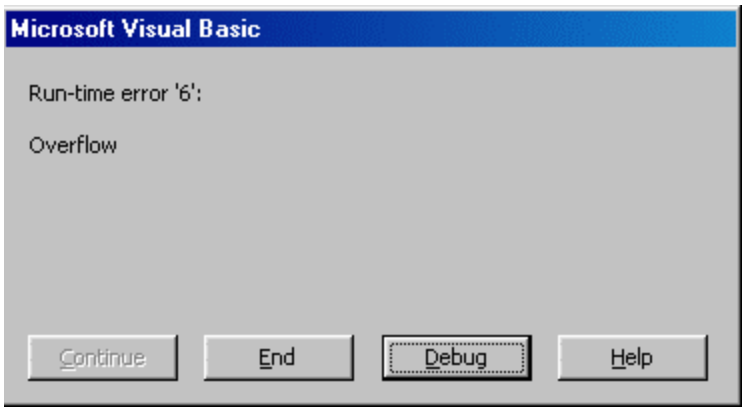
Имя_Проекта - Microsoft Visual Basic [**break**]

Давайте изменим код:

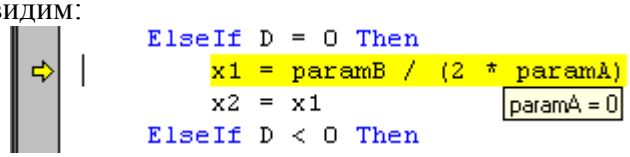
```
paramA = txtParamA.Text
paramB = txtParamB.Text
paramC = txtParamC.Text
```

На код:

```
paramA = Val(txtParamA.Text)
paramB = Val(txtParamB.Text)
paramC = Val(txtParamC.Text)
```



Нажимаем Debug и вот что видим:



Причина ошибка заключается в невозможности деления на 0, а paramA 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 paramA = 0 Or paramB = 0 Or paramC = 0 Then
    MsgBox "Нули в качестве коэффициентов не допускаются!", _
    vbCritical
    Exit Sub
End If
```

Символ "_" используется в том случае, когда вы хотите перенести часть выражения на другую строчку. В данном случае мы переносим константу vbCritical.
Здесь мы проверяем переменные на содержание в них нулей. В принципе, можно было бы проверить не переменные, а сами тектовые поля (If Val(txtParamA.Text)=0 Then....). Это уже дело вкуса. Всё равно, результат одинаков.

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. Такие ошибки называются логическими. Здесь я бы хотел привести очень интересный и поучительный отрывок из книги "Программирование в среде Visual Basic 5":
"Прежде чем отлаживать программу, необходимо убедиться в том, что она содержит ошибки. Поэтому тестирование программы является первым шагом на пути её отладки. Вот именно такая логическая ошибка оставила клиента без электричества. Для нахождения этих ошибок очень удобно использовать пошаговую трассировку.
. Нажмите правой кнопкой мыши на той строчке кода, где вы хотите поставить брикпоинт и в меню выбрать Toggle->Breakpoint. Замечание: такую точку нельзя ставить на строчке с объявлением переменной.

Обработка других ошибок

Независимо от того, насколько качественно написано приложение, никогда нельзя полностью исключать возможность возникновения ошибки в программе. Вы видели такое окошко Windows: "Программа выполнила недопустимую операцию и будет закрыта...". Т.е. возникла исключительная ситуация, и Windows, не зная как можно обойти ошибку, выдаёт "общее" окно для всех таких ошибок. Так вот и в ваших программах тоже может возникнуть такая исключительная ситуация. Причин для этого ооочень много. Например, у пользователя программой сбои с жёстким диском, или глючит операционная система, или вирус удалил нужный вам файл и т.д. и т.п. В общем, в таких ситуациях желательно сообщить пользователю о том, что возникла ошибка и спокойно выйти/продолжить программу.. В данном случае используется объект Err.
Visual Basic располагает оператором, с помощью которого можно контролировать ход программы при возникновении ошибок. Это оператор On Error. Он имеет несколько видов:

```
On Error GoTo МЕТКА
On Error Resume Next
On Error GoTo 0
```

Первый оператор позволяет указать Visual Basic метку (номер строки) на которую передастся управление программы при возникновении ошибки. Рассмотрим пример использования этого оператора:

```
Private Sub Command1_Click()
    Dim myString As String
    On Error GoTo ERRH
    myString = "ERROR HANDLING WITH VB IS COOL"
    MsgBox Mid(myString, 0, 1) '<--- здесь ошибка (#)
    Exit Sub '<--- досрочно выходим, если нет ошибок
ERRH: '<--- метка
    MsgBox Error(Err.Number) '<--- выводим свою ошибку
End Sub
```

Урок № 17. Доводим до ума

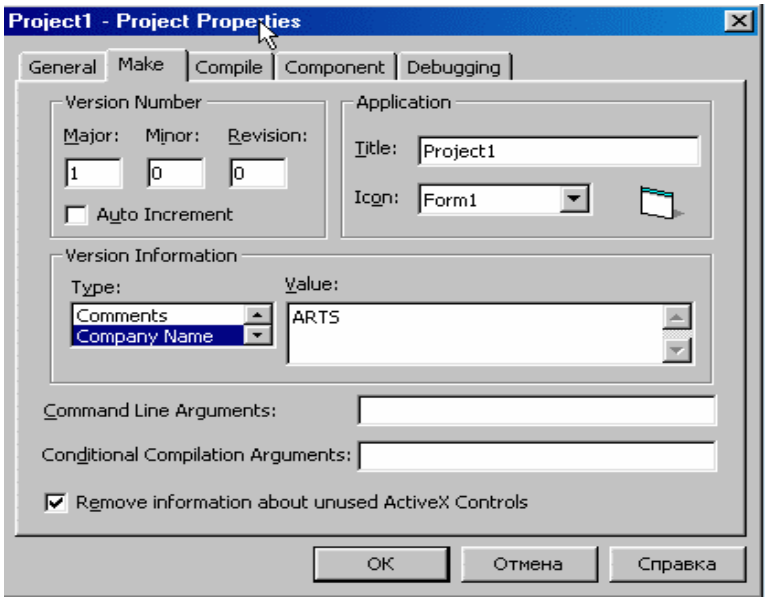
На этом уроке мы научимся оформлять программу в виде функций и процедур. Это очень важно. Принцип модульного программирования очень облегчает программирование и отладку. Модульное программирование - означает разделение кода программы на отдельные куски, каждый из которых выполняет чётко определённую задачу. Это особенно важно для сложных программ.
Доведём до ума нашу программу, для вычисления корней квадратного уравнения. Давайте оформим некоторые части кода в виде процедур и функций. Например, напишем функцию для вычиления дискриминанта. И напишем процедуру, которая будет выводить полученные значения в метки. Почему именно процедуру? Потому что процедуры пишутся для выполнения некоторой последовательности действий, где не требуется возвращать какое-либо значение. Давайте ещё раз посмотрим на ту часть кода, где происходит проверка значения дискриминанта и вычисление корней уравнения:

```
If D > 0 Then
    x1 = (paramB + Sqr(D)) / (2 * paramA)
    x2 = (paramB - Sqr(D)) / (2 * paramA)
    lblD.Caption = "Дискременант: " & D
    lblX1.Caption = "Корень №1: " & x1 ' <- здесь
    lblX2.Caption = "Корень №2: " & x2
ElseIf D = 0 Then
    x1 = paramB / (2 * paramA)
    x2 = x1
    lblD.Caption = "Дискременант: " & D
    lblX1.Caption = "Корень №1: " & x1 ' <- и здесь
    lblX2.Caption = "Корень №2 = Корню №1"
ElseIf D < 0 Then
    lblD.Caption = "Дискременант: " & D
    lblX1.Caption = "Корней нет!" ' <- и здесь, тоже
    lblX2.Caption = ""
    MsgBox "Дискременант меньше нуля! Корней нет!", vbCritical
End If
```

Заметьте, что в каждом из ветвей оператора If наблюдается присвоения свойству Caption 3-х меток (lblD, lblX1, lblX2). Поэтому логично этот участок оформить в виде процедуры, параметрами которой будут значения для меток.

Урок № 18. Компиляция

Теперь наша программа отлажена и готова к употреблению. Теперь настало время узнать, как же откомпилировать программу в exe файл?
Напомню, что Visual Basic предлагает 2 компилятора. Компиляция в P-код, и компиляция в Native-код. P-код - это старый компилятор и пользоваться им я не рекомендую. Всегда компилируйте приложение в Native-код. Выбор вида компиляции находится на вкладке Compile в меню Project->Project Properties. Там же можно указать несколько доступных видов оптимизации (о них ниже).



Итак, чтобы откомпилировать нашу программу, необходимо проделать следующие манипуляции:

1. В меню File выбрать Make имя_проекта.exe
2. Ввести имя выходного exe файла
3. Если необходимо, то выбрать некоторые опции, нажав на кнопку Options. (К этим опциям также можно добраться через меню Project->Project Properties).
4. Нажать OK. И, если Visual Basic не найдёт никаких ошибок в программе, то откомпилирует её и сохранит в указанно вами каталоге под указанным вами именем.

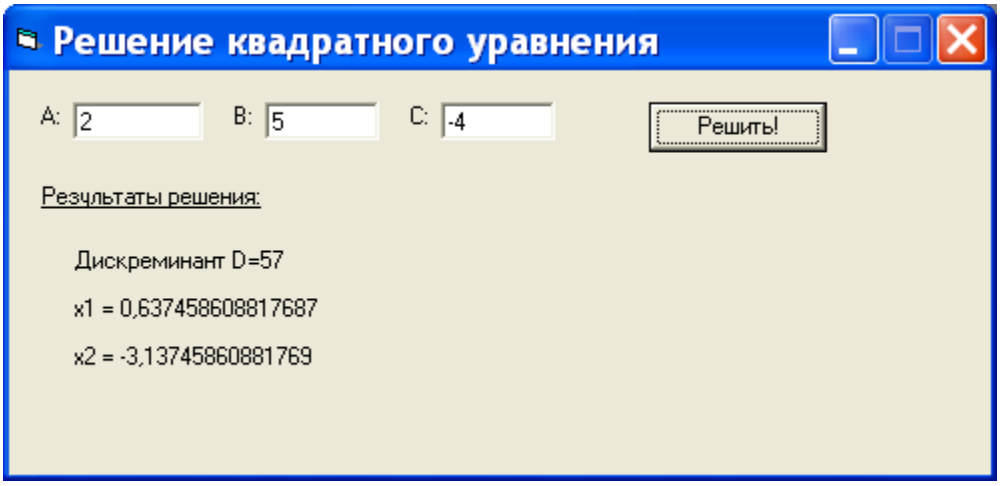
Заключение

Вот мы и подошли к концу этого небольшого курса для начинающих. Мы рассмотрели основные принципы программирования на Visual Basic.
Успехов вам в программировании!

Литература:

1. Сайт http://vb.hut.ru
2. А. Волков - Электронный учебник «**Visual Basic – крепкий орешек!».**
3. Н. Угринович - Информатика и информационные технологии. 10-11 кл. М. БИНОМ, 2006.

Модель «Решение квадратного уравнения»




```
Private Sub cmdCalculate_Click()
   'объявляем переменные
   Dim paramA As Double
   Dim paramB As Double
   Dim paramC As Double
   Dim R As Integer
   Dim D As Double
   Dim x1 As Double
   Dim x2 As Double

   'присваиваим им значения из текстовых полей
   paramA = Val(txtParamA.Text)
   paramB = Val(txtParamB.Text)
   paramC = Val(txtParamC.Text)

   'проверка на a=0
   If paramA = 0 Then
   MsgBox "A=0 не допускается!", vbCritical
   Exit Sub
   End If

   'вычисляем дискреминант
   D = paramB * paramB - 4 * paramA * paramC
   'решаем уравнение и выводим результат в метки Label
   If D > 0 Then
      x1 = (-paramB + Sqr(D)) / (2 * paramA)
      x2 = (-paramB - Sqr(D)) / (2 * paramA)
      lblD.Caption = "Дискреминант D=" & D
      lblX1.Caption = "x1 = " & x1
      lblX2.Caption = "x2 = " & x2
   ElseIf D = 0 Then
      x1 = -paramB / (2 * paramA)
      lblD.Caption = "Дискреминант D=" & D
      lblX1.Caption = "x=" & x1
      lblX2.Caption = ""
   Else
      lblD.Caption = "Дискреминант D=" & D
      lblX1.Caption = "Нет решений"
      lblX2.Caption = ""

      'можно добавить выводное поле с предупреждающей информацией
      R = MsgBox("Дискреминант меньше нуля! Корней нет!", 48, "Внимание!")
   End If
End Sub
```

Программа (*программный код*)**:**

```
Dim I As Integer
Dim X As Double
Dim Y As Double
Private Sub cmdCK_Click()
picGrap.Scale (-10, 5)-(10, -5)
picGrap.Line (-10, 0)-(10, 0)
For I = -10 To 10
picGrap.PSet (I, 0)
picGrap.Print I
Next I
picGrap.Line (0, -5)-(0, 5)
For I = -5 To 5
picGrap.PSet (0, I)
picGrap.Print I
Next I
End Sub
Private Sub cmdCLS_Click()
picGrap.Cls
End Sub
Private Sub cmdSin_Click()
For X = -10 To 10 Step 0.03
Y = Sin(X)
picGrap.PSet (X, Y)
Next X
End Sub
Private Sub cmdCos_Click()
For X = -10 To 10 Step 0.03
Y = Cos(X)
picGrap.PSet (X, Y)
Next X
End Sub
Private Sub cmdTg_Click()
For X = -10 To 10 Step 0.03
Y = Tan(X)
picGrap.PSet (X, Y)
Next X
End Sub
Private Sub cmdCtg_Click()
For X = -10 To 10 Step 0.03
Y = 1 / Tan(X)
picGrap.PSet (X, Y)
Next X
End Sub
```

Имеется возможность очистки экрана и построения графиков по одному, по два, по три или все четыре. А также замены функций на другой вид, до четырех видов, возможность графического решения уравнений и неравенств.