Оглавление

Visual Basic — средство разработки программного обеспечения, разработанное корпорацией Microsoft и включающее язык программирования и среду разработки. Язык Visual Basic унаследовал дух, стиль и отчасти синтаксис своего предка — языка Бэйсик, у которого есть немало диалектов. В то же время Visual Basic — современный язык программирования, сочетающий процедуры и элементы объектно-ориентированных и компонетно-ориентированных языков программирования. Среда разработки VB включает инструменты для визуального конструирования пользовательского интерфейса.

Стандартный проект на Visual Basic

Приложение, создаваемое в среде Visual Basic, называется проектом.
Программный проект — это совокупность частей, составляющих будущее WINDOWS-приложение. Любой проект должен обязательно состоять из экранных форм (хотя бы одной) и программных модулей (хотя бы одного). Visual Basic хранит каждый проект в отдельном файле с расширением vbp.
Программный модуль — это хранящийся в отдельном файле программный код (текст некоторой программы). Он может использоваться при решении чаще всего одной, а иногда и нескольких задач. Имя этого файла имеет расширение bas. Программный код проекта существует не сам по себе, он привязан к отдельным объектам экранной формы. Часть кода, которая относится только к одному объекту, в свою очередь может состоять из нескольких фрагментов-процедур.
Разработка интерфейса программы выполняется с помощью конструктора форм.
Экранная форма — это графическое представление WINDOWS-приложения вместе с содержанием этого окна. Содержание включает в себя:

- совокупность свойств этого окна с их значениями;
- совокупность, объектов, находящихся в этом окне;
- совокупность свойств этих объектов с их значениями.

В Visual Basic экранная форма хранится в отдельном файле с расширением frm.
Чтобы программа выполнялась, исходные тексты переводят на машинный язык. Это делает компилятор, который также водит в систему программирования.

Выбор типа проекта

При вызове Visual Basic открывается окно диалога мастера проектов. Оно имеет 3 вкладки следующего назначение:
New – создание нового проекта, предлагается на выбор несколько стандартных шаблонов, для создания Вашего первого приложения выбираем **Standart EXE**.
Existing – открыть существующий проект, позволяет выбрать файл в диалогом окне выбора.
Recent – открыть один из последних проектов, которые были созданы или в которые были внесены любые изменения.

Основные понятия в VB.

Имена (идентификаторы) - последовательность символов для обозначения объектов программы (переменных, массивов, функций и др.).
Имя объекта (идентификатор) — это строка символов, которая отличает эту переменную от других объектов программы (идентифицирует переменную в программе).

При задании имен нужно соблюдать следующие правила:

- первым символом имени должна быть буква. Остальные символы — буквы и цифры (прописные и строчные буквы различаются).
- имена могут содержать только букы, цифры и символ подчеркивания;
- число символов в имени не должно превышать 255;
- имя не должно совпадать с зарезервированными (служебными) словами языка.

Данные - величины, обрабатываемые программой. Имеется три основных вида данных: **константы**, **переменные** и **массивы**.

- **Константы** - это данные, которые зафиксированы в тексте программы и не изменяются в процессе ее выполнения.
  Константы — величины, значения которых не могут меняться. Как и переменные, константы объявляются в начале текста программного кода оператором:

  Const ИмяКонстанты [As Тип] = Значение

  например: Const As Double Pi = 3.14159

- **Переменные** – это данные, которые могут изменять свои значения в ходе выполнения программы. Они обозначаются именами.

Образно их можно представить в виде "ящика", имеющего некое содержимое, например, символы или числа. Этому "ящику" присваивается имя, т. е. имя переменной. Чтобы сослаться на содержимое, достаточно указать имя переменной. В зависимости от содержимого различают переменные разных типов (целые числа, с плавающей запятой, строки и т. п.).
Переменная — это именованное хранилище данных определенного типа.
Переменная — это именованная область памяти, предназначенная для хранения данных, изменяющихся в процессе выполнения программы. Для доступа к содержимому этой области памяти достаточно знать имя переменной.

- **Массивы** - последовательности однотипных элементов, число которых фиксировано и которым присвоено одно имя. Положение элемента в массиве однозначно определяется его индексами - одним в случае одномерного массива, или несколькими, если массив многомерный.
    - Их ещё называют списками. Массив (вектор) - это набор однотипных переменных, объединенных одним именем и доступных через это имя и порядковый номер переменной в наборе. Элементы массива обладают непрерывной нумерацией определённого диапазона.

Общий синтаксис определения массива следующий:
Dim ИмяМассива (НомПерв1 To НомПосл1, НомПерв2 To НомПосл2, ...) [As [New] ИмяТипа]
Пример объявления массива

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

Как вы могли заметить, определение массива отличается от определения обычной переменной только индексом, указанным в скобках. Этот индекс указывает размерность массива. В данной случае массив myArray будет содержать 11 элементов. Почему 11? Потому что нижняя граница массива начинается с нуля. [0,1,2.....9,10]. Чтобы задать определённую размерность можно использовать зарезервированное слово To:

```
Dim myArray (5 To 10) As Long
```

Здесь определяется массив, размерность которого 6 элементов (5,6,7,8,9,10).
Многомерные массивы
Массивы можно делать многомерными. Например, объявим массив - таблицу поля шахматной доски:

```
Dim chessTable (1 To 8, 1 To 8) As String
```

Этот массив представляет собой таблицу с восьми ячейками по вертикали и горизонтали.
Итак, массив определён. Теперь необходимо узнать - как же можно добраться к элементам этого массива. Очень просто! К элементам массива нужно обращаться по индексу, к примеру, чтобы изменить нулевой элемент массива myArray нужно написать:

```
myArray(0) = 1234
Или, например:
chessTable (2,3) = "Пешка"
```

Типы данных в языке Visual Basic

1. **Byte** — целое неотрицательное число. Принимает значения от 0 до 255, занимает 1 байт.
2. **Boolean** — логическая величина. Принимает одно из двух значений True и False, занимает 2 байта.
3. **Currency** — десятичное протяженное число. Используется для проведения денежных вычислений с фиксированным количеством знаков после десятичной запятой, занимает 8 байт. (До 15 знаков слева от десятичной точки и до 4 знаков справа от десятичной точки).
4. **Date** — дата. Используется для хранения дат/времени, занимает 8 байт.
5. **Double** — десятичное число двойной точности, занимает 8 байт.
6. **Integer** — короткое целое числовое значение, занимает 2 байта. Используется для представления целых чисел в диапазоне от -32768 до 32767.
7. **Long** — длинное целое число, занимает 4 байта. Используется для представления целых чисел в диапазоне от -2147483648 до 2147483647.
8. **Single** — десятичное число обычной точности, занимает 4 байта. Используется для представления отрицательных чисел в диапазоне от -3.402823E 38 до -1.401298E-45 и для представления положительных чисел в диапазоне от 1.401298E-45 до 3.402823E 38.
9. **String** — строка переменной длины. Занимаемая память линейно зависит от числа символов в строке.
10. **String *** длина — строка постоянной длины.

11. **Variant** — произвольное значение. Используется для хранения данных любых типов. Занимает 16 байтов плюс по одному байту на каждый символ, если значением является строка. Использование такого типа данных, как variant, замедляет работу программы, так как требуется время и ресурсы для операций преобразования типов
12. **Object** — объект. Используется только для хранения ссылок на объекты, занимает 4 байта.

Иногда, при использовании некоторых операторов, API-функций, а также просто для собственных нужд необходимо создавать собственные типы данных. Часто их называют структурами. По своей сути структура - это как бы одномерный массив, который мы запихиваем в одну переменную. Но в него могут входить данные разных типов.

Возможности объявления типа переменной в Visual Basic

- **Переменная может вообще не объявляться.** В этом случае будет установлен по умолчанию тип Variant. Однако это может привести к медленной неэффективной работе программы и нерациональному использованию памяти.
- **Переменная может объявляться явно с помощью оператора определения переменной:**

  Dim имяПеременной [As типПеременной],
  например: Dim d as Long.

- **Переменная может объявляться неявно:**
  - с помощью оператора объявления типа:
    DefТип Начальная буква [— Конечная буква];
    вместо ОегТип указывается одно из ключевых слов, обозначающих тип:

    — DefInt (для типа Integer);
    — DefLng (для типа Long);
    — DefSng (для типа Single);
    — DefStr (для типа String) и так далее;

  - о типе переменной можно иногда судить по суффиксу, приписываемому к имени переменной. Суффиксы могут быть только у шести типов переменных:
    % — тип Integer;
    & — тип Long;
    ! — тип Single;
    # — тип Double;
    $ — тип String;
    @ — тип Currency.

Dim Price As Currency Price = 100 или Price@=100

Если переменная, тип которой указан неявно с помощью суффикса, встречается в программе многократно, то суффикс можно указывать только один раз при первом использовании этой переменной в программе.

Примеры объявления переменных
Dim a As Integer - объявлена переменная а целочисленного типа.
Dim t ,f As Double - объявлены переменные t и f с плавающей точкой.
Dim s As String - объявлена переменная s - строка.

В зависимости от типа переменной, выполняются различные действия

Пример 1

```
Dim a, b, c As Integer
        a = 5
        b = 3
        c = a + b
```

При таком объявлении переменных, переменная «с» будет равна 8

Пример 2

```
Dim a, b, c As String
        a = 5
```

```
        b = 3
        c = a + b
```
При таком объявлении переменных, переменная «c» будет равна 53

Пример 3 – требуется поменять значения переменных a и b
```
Dim a, b, c As Integer
        a = 5
        b = 8
       'Для обмена используем вспомогательную переменную c
            c=a
            a = b
            b = c
```

после этого a будет равно 8, b 5

Пример 4 объявляем массив из 6 элементов (нумерация начинается с 0)
```
Dim A(5) As Integer
```

A	0	1	2	3	4	5
	5	8				

A(0)=5
Запишет 5 в ячейку 0.
A(1)=8
Запишет 8 в ячейку 1.

Пример 5 объявления массива из 36 элементов (нумерация начинается с 0)
```
Dim A(5, 5), t As Integer 'Объявляем массив, для записи целочисленных
значений
```
A

	0	*1*	*2*	*3*	*4*	*5*
0	8					
1		5				
2						
3						
4						
5						

```
A(1, 1) = 5
'Запишет  5  в  ячейку  1,1.
A(0, 0) = 8
'Запишет  8  в  ячейку  0,0.
Label1.Text = A(1, 1)
t=A(1,1)     'Присвоит переменной t значение 5.
```

Пример 6
```
Dim A(5, 5) As String 'Объявляем массив, для записи строк
```
A

	0	*1*	*2*	*3*	*4*	*5*
0	F					
1		B				
2						
3						
4						
5						

```
A(1, 1) = "B"
'Запишет  B  в  ячейку  1,1.
A(0, 0) = "F"
'Запишет  F  в  ячейку  0,0.
```

Не используйте имена элементов управления по умолчанию.

Этот совет имеет непосредственное отношение к пункту об осмысленных именах переменных. Если будете использовать имена по умолчанию, то вы очень быстро запутаетесь какое из 15 TextBox с номерами отвечает например за ввод имени пользователя.
Плохо: Form1, Textbox2, ListView15.
Хорошо: frmMain, txtName, lvwListOfCommands.

Используйте префиксы типов.

Префиксы типов (так называемая венгерская нотация) - это определённые символы в начале имени переменной, которые в закодированном виде несут информацию о переменной. Общепринятыми являются следующие префиксы:
s или sz - переменная типа String
b - Byte или Boolean
i - Integer
l - Long
d - Double
s - Single
d - Date
o_ - объект
e - перечислимый тип (Enum)
g_ - глобальная переменная (Public)
m_ или loc_ - локальная переменная (Private)
и т.д.
Для элементов управления на форме:
txt - TextBox
pic - PictureBox
lbl - Label
fr или fra - Frame
tvw - TreeView
и т.д.
Для модулей:
frm - форма
mod - модуль
cls - модуль класса

Капитализация vs подчёркивание.

В случаях когда имя идентификатора состоит из нескольких слов для удобства чтения их следует как-то разделить. Для этого есть два основных способа.
Капитализация - первая буква каждого слова в имени переменной записывается с заглавной буквы, остальные строчными.
Подчёркивание - слова разделяются символами подчёркивания "_".
Плохо: filenameslist, sfilenameslist
Хорошо: filenames_list, FileNamesList, sFileNamesList.

Пишите комментарии.

Старайтесь по возможности писать комментарии. Если вы покажете кому-то свой код, то ему будет намного проще сориентироваться в нём, если там будут подсказки в виде комментариев. Даже если вы не собираетесь ничего никому показывать всё равно следует писать комментарии. Так как просматривая свой код уже через месяц вы будете с трудом вспоминать что и как у вас работает и зачем нужен тот или иной кусок. Комментарии выступают как своеобразные записки самому себе на будущее.

Хорошим тоном является перед каждой значимой подпрограммой (процедурой, функцией, методом, свойством) писать небольшой заголовок с описанием подпрограммы: что она делает, входные параметры, возвращаемые значения, особенности её использования и прочие замечания. Например вот в таком виде:

Код:

```
Dim a,b,c as Integer                    ' Расчёт суммы
' рассчитаем  значение c
c=a+b
```

Выражения – элементы языка, которые предназначаются для выполнения необходимых вычислений, состоят из констант, переменных, указателей функций, объединенных знаками операций. Выражения записываются в виде линейных последовательностей символов (без подстрочных и надстрочных символов, "многоэтажных" дробей и т. д.), что позволяет вводить их в компьютер, последовательно нажимая на соответствующие клавиши клавиатуры.
Различают выражения **арифметические**, **логические** и **строковые**.
Существуют следующие типы выражений:

- **Арифметические выражения** служат для определения одного числового значения. Арифметические выражения записываются по следующим правилам:
  1. Нельзя опускать знак умножения между сомножителями и ставить рядом два знака операций.
  2. Индексы элементов массивов записываются в скобках.
  3. Операции выполняются в порядке старшинства: сначала вычисление функций, затем возведение в степень, потом умножение и деление и в последнюю очередь - сложение и вычитание.
  4. Операции одного старшинства выполняются слева направо.

  арифметические операторы:
  - ^ оператор возведения в степень.
  - * оператор умножения.
  - / оператор деления
  - \ оператор целочисленного деления
  - Mod оператор вычисления остатка от деления
  - + оператор сложения
  - - оператор вычитания
- **Логические выражения** описывают некоторые условия, которые могут удовлетворяться или не удовлетворяться. Таким образом, логическое выражение может принимать только два значения - "истина" или "ложь" (да или нет).
  В записи логических выражений помимо арифметических операций сложения, вычитания, умножения, деления и возведения в степень используются операции отношения и логические операции.

  логические операторы:
  - AND- операция логическое И или логическое умножение (конъюнкция). И одно, и другое условные выражения должны быть истинны, чтобы все сложное выражение можно было считать истиной.
  - OR- операция логическое ИЛИ или логическое сложение (дизъюнкция). Достаточно, чтобы одно из выражений было истинным, чтобы все сложное выражение было истинным.
  - XOR операция исключающее ИЛИ. Обычное ИЛИ дает true, когда оба операнда true, а данный вариант исключает по принципу или-или, но не оба вместе и дает false. Итак, если одно и только одно условное выражение имеет значение ИСТИНА, то результат будет ИСТИНА. Если оба условия ИСТИНА или оба ЛОЖЬ, то результат будет ЛОЖЬ.
  - NOT- операция "логическое НЕ" или отрицание. Это операция с одним операндом. Если операнд является истинным, то все выражение- будет ложью. И наоборот.

  отношения: меньше, обозначается символом “<”; больше, обозначается символом “>”; меньше или равно, обозначается символами “<=”; больше или равно, обозначается символами “>=”; равно, обозначается символом “=”; не равно, обозначается символами “<>”.
- **Строковые выражения**
  - **конкатенации** символьных значений друг с другом, изображается знаком "&". Оператор конкатенации строк (&) объединяет две строки в одну, но не через операцию сложения. Результатом является комбинация символа "9" и символа "2". Конкатенация строк может выполняться с числовыми переменными - например, если вы отображаете счет бейсбольного матча так, как это делается в старых полях для отображения счета - но гораздо чаще конкатенация используется для строковых значений или переменных.
  - **оператор сложения** в конкатенации строк. Для конкатенации строк можно также использовать оператор (+). Этот оператор имеет такой же синтаксис и требования, как и оператор (&). Однако следует понимать, что в VBA основное предназначение оператора (+) - это арифметическое сложение. Поэтому, чтобы избежать двусмысленности чтения программного кода, для конкатенации строк настоятельно рекомендуется использовать именно оператор (&).

- **сравнение строк.** При сравнении строк операторами отношения, VB сравнивает каждую строку слева направо посимвольно. В VB одна строка равна другой только, когда обе строки содержат точно такие же символы в точно таком же порядке и обе строки имеют одну и ту же длину.

Ключевые слова – это слова языка, имеющие строго определенное назначение, которые не могут использоваться в качестве идентификаторов.

Оператор – это элемент языка, который задает полное описание некоторого действия, которое необходимо выполнить. Оператор - это наиболее крупное и содержательное понятие языка: каждый оператор представляет собой законченную фразу языка программирования и определяет некоторый вполне законченный этап обработки данных. В состав операторов входят ключевые слова; данные; выражения и т.д.

Управляющие структуры

Условия

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 случает оператор может быть только один. Во втором сколько угодно (в том числе и один).

Пример 1:

```
Dim a, b, c As Integer
        a = 5
        b = 3
        If (a > b) Then
            c = a - b
        End If
```

Пример 2:

```
   Dim a, b, c As Integer
        a = 5
        b = 3
      If (a > b)And (a>c) Then
         c = a - b
        End If
```

Скобки здесь не обязательны, но они повышают читабельность кода.

Условный оператор If...Else...ElseIf...End If

Такая конструкция используется для более сложных ветвлений:

```
If Логическое_выражение 1 Then
   Группа_операторов
ElseIf Логическое_выражение 2 Then
   Группа_операторов
...
Else
   Группа_операторов
End If
```

Эта схема может быть и в укороченном виде If...Then...Else...End If. При этом операторы после Else выполняются только в том случае, если ни одно из условий не выполнено.

Пример 1:

```
Dim a, b, c As Integer
        a = 5
        b = 3
        If (a > b) Then
            c = a - b
```

```
        Else
            c = b - a
        End If
```

Пример 2: Нахождение корней квадратного уравнения

```
Dim a, b, c As Integer
Dim d,x1,x2 As Double
        a = 5
        b = 3
        c=1
        d=b^2-4*a*c     'Находим дискриминант
        If (d >0) Then
            x1=(-b+math.Sqrt(d))/(2*a)
            x2=(-b-math.Sqrt(d))/(2*a)
            Label1.Text = “x1=” + Str(x1)+ ”  “ + “x2=” + Str(x2)
        ElseIf (d=0)  Then
            x1=(-b+math.Sqrt(d))/(2*a)
            x2=“один корень”
            Label1.Text = “x1=” + Str(x1)+ ”  “ + “x2=” + Str(x2)
        Else
          Label1.Text = “нет корней”
        End If
```

Циклы

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 1:

В этом примере всем элементам массива iArray присваивается значение 5.

```
Dim iArray(10) As Integer
        For i = 0 To 10     ' Пробегаем по всем элементам массива
            iArray(i) = 5  'присваеваем элементу 5
        Next i              ' переход к следующему элементу
```

Пример 2:

Заполняем случайными значениями от 0 до 100 массив **iArray**

```
Dim iArray(10) As Integer
        For i = 0 To 10     ' Пробегаем по всем элементам массива
            iArray(i) = Rnd() * 100  ' Rnd() возвращает значение случайное
значение от 0 до 1 (например 0,564425), домножаем его на 100 и получим 56.
            Label1.Text = Label1.Text + " " + Str(iArray(i)) ' Выводим в
надпись все значения массива
        Next i
```

Пример 3:

Заполняем значениями 5 все четные элементы массива и -1 нечетные элементы массива
Это можно сделать несколькими способами

Вариант 1

```
Dim iArray(10) As Integer
        For i = 0 To 10 ' Пробегаем по значениям от 0 до 10
            If (i Mod 2 = 0) Then ' Mod - возвращает остаток от деления
                iArray(i) = 5     'присваеваем элементу 5
            Else
```

```
                iArray(i) = -1    'присваеваем элементу -1
            End If
            Label1.Text = Label1.Text + " " + Str(iArray(i)) ' Выводим в
надпись все значения массива
        Next i
```

Вариант 2

```
Dim iArray(10) As Integer
        For i = 0 To 10 Setp 2 ' Пробегаем по значениям от 0 до 10 с шагом 2
           iArray(i) = 5     'присваеваем элементу 5
        Next i
        For i = 1 To 10 Setp 2 ' Пробегаем по значениям от 1 до 10 с шагом 2
           iArray(i) = -1     'присваеваем элементу -1
        Next i
        For i = 0 To 10 ' Пробегаем по значениям от 0 до 10
          Label1.Text = Label1.Text + " " + Str(iArray(i)) ' Выводим в
надпись все значения массива
        Next i
```

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).

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
```

второй вариант

```
Do
   Группа операторов
Loop While Условие_выхода
```

Отличие между ними заключается в том, что условие выхода проверяется в одном случае перед очередным проходом, а в другом случае - после выхода. Если в цикле опустить условие выхода или это условие всегда выполняется, то получится бесконечный цикл. Например вот такой

```
Do While 2 > 1
   Degub.Print "Вечный цикл"
Loop
```
Если у вас случаянно получился такой цикл, то выйти из него можно при нажатии Ctrl+Break. Но это работает только в среде разработки.

Пример:
```
Dim n As Integer
n = 100
Do While n >= 0
   n = n - 1
   Debug.Print n
Loop
```

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
```

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 таких использования таких структур не получится написать даже самую маленькую программу. Даже если и получится, то программа не будет представлять никакого практического интереса.

Функции VB

Mod – арифметическая операция деления с остатком. Этот оператор возвращает остаток от деления делимого на делитель целое количество раз. Если делитель и делимое являются целыми типами, возвращенное значение является целым. Если делитель и делимое являются типами с плавающей запятой, возвращенное значение также является переменной с плавающей запятой. Следующий пример иллюстрирует такое поведение.
```
Dim x,y,z As Integer
z = x Mod y   ' Mod – возвращает остаток от деления
```

Val -Возвращает числа, содержащиеся в строке в качестве числовых значений соответствующего типа. Пример.
```
Dim x As Integer
Dim s As String
s=”54app”
x=Val(s)    ' преобразует строку в число x равен 54
```

Str- Представляет возвращаемое числовое значение как строку. Пример.

```
Dim x As Integer
Dim s As String
x=5
s=Str(x)    ' преобразует число в строку
```

Процедуры и функции

Процедуры и функции представляют собой отдельные блоки, из которых складывается код программы, каждая процедура выполняет какую-то задачу или ее часть.
Процедуры обработки событий после вызова постоянно находятся в ожидании событий.

Кроме процедур обработки событий в программу можно включить процедуры и функции не связанные с событиями. Они выполняют отдельные действия и могут быть использованы неоднократно. Назовем их общими. Процедуры общего назначения вызываются на выполнение в коде программы. Использование процедур экономит время и позволяет избежать лишних ошибок. ***Функции отличаются от процедур тем, что возвращают какое-то значение.***

Подпрограмма - поименованная или иным образом идентифицированная часть компьютерной программы, содержащая описание определённого набора действий. Подпрограмма может быть многократно вызвана из разных частей программы.

Под процедурой или функцией часто понимается последовательность операций, которую нужно многократно выполнять в различных местах приложения. При этом требуемый блок команд записывается в коде только один раз, после чего к нему можно обращаться из любой части программы.

Функция – это подпрограмма, которую вызывают, чтобы выполнить какие-то расчеты или проверки. Когда она завершает работу, то возвращает управление вызывающей программе и передает ей результат расчета.

Процедура – это тоже подпрограмма. Ее тоже вызывают, чтобы выполнить какие-то действия, но от нее не требуется возвращать основной программе какие-либо значения.

Синтаксис объявления процедуры и функции:

Код:

```
[Public/Private][Static] Sub <Имя процедуры>(<Параметры>)
<Операторы>
End Sub

Function <Имя функции> [As тип]
<Операторы>
End Function
```

Процедуры, объявленные с ключевым словом **Public**, можно вызвать в любом модуле приложения (каждая форма – это отдельный модуль).

Процедуры объявленные как Private, можно вызывать только в текущем модуле.

Слово Static означает, что все переменные, объявленные в процедуре, будут статическими, т.е. их значения сохраняются между вызовами.

Параметры обеспечивают связь процедуры с приложением. Это данные, передаваемые в процедуру при вызове.

Процедуры обработки событий. Вызываются в том, случае если произошло какое-либо событие. При этом существенным является как имя элемента, та и вид события, который с ним произошел.

Пользовательские процедуры. Группы операторов, создаваемые разработчиком для выполнения определенных задач и не зависящие от текущего состояния приложения или произошедших в тот

или иной момент событий.

Встроенные функции. Определенные наборы команд, имеющиеся в языке Visual Basic и в предназначенные для вычисления тех или иных значений на основании исходных данных. Встроенными являются, в частности, как математические, так и строковые функции (Abs, Cos, Sin, Mid, Len и т.д.)

Пользовательские функции. Группы операторов, аналогичные пользовательским процедурам.

Однако между ними есть ряд отличий.
Основные отличия функции от процедуры состоят в следующем.
1. Функция имеет тип (аналогично переменной) и может возвращать в программу значение, которое присваивается функции при помощи оператора:
<Имя функции> = значение
2. Вызов функции, как правило, осуществляется посредством указания в правой части какого-либо оператора ее имени и параметров. С другой стороны, процедура вызывается при помощи отдельного оператора:

Call <Имя процедуры> (Параметры)
Или
<Имя процедуры> (Параметры)

Если при вызове процедуры используется ключевое слово Call, то список параметров должен быть указан в скобках. Если же процедура вызывается без использования Call, то ее параметры перечисляются без скобок.

Необходимо отметить, что вызываемая процедура может не иметь параметров. В этом случае (если использовалось служебное слово Call) после имени процедуры следует ставить пустые скобки.

Пользовательские процедуры обычно используются при необходимости выполнения одной и тоже последовательности операций. Например, в программе требуется неоднократно вводить в цикле значения массива arrA, состоящего из пяти элементов. В этом случае заполнение массива лучше всего оформить в виде процедуры.
Команда Add Procedure меню Tools позволяет добавить процедуру или функцию.
Пусть процедура CountSumm вычисляет сумму a и b, которые передаются в процедуру как параметры. Создавая процедуру CountSumm командой Add Procedure, нужно указать имя процедуры и выбрать область видимости Public или Private.

Теперь нужно вписать параметры в скобки и написать текст процедуры. В списке параметров рекомендуется указывать тип переменных.

Код:

```
Private Sub CountSumm(a As Integer, b As Integer)
c = a+b
End Sub
```

```
Function CountSumm (a As Integer, b As Integer)  As Integer
c=a+b
End Function
```

Процедуры начинаются оператором Sub и заканчиваются оператором End, между которыми и помещается код. Они могут вызываться или самим Visual Basic (процедуры обработки событий), или другими процедурами. Имя процедуры обработки события состоит из имени объекта и имени события:
Private Sub Commandl Click()
End Sub