СОДЕРЖАНИЕ

1 ВВЕДЕНИЕ

Microsoft Office – это среда, в которой решение многих задач не требует знания программирования. В то же время, создатели пакета Microsoft Office предусмотрели прекрасную возможность для пользователя – возможность самому конструировать профессиональные приложения, работающие в любом из компонентов пакета Microsoft Office. Это существенно расширило возможности применения пакета Microsoft Office в различных сферах деятельности и бизнеса. Для создания собственных приложений пользователю предоставлен программный инструментарий, встроенный в пакет Office – редактор языка Visual Basic for Applications (VBA). Встроенный язык открывает дорогу достаточно новому направлению в современном программировании – офисному программированию.

VBA является одним из самых популярных языков программирования, прост в освоении и позволяет быстро получить ощутимые результаты. VBA применяет технологию визуального программирования, позволяющую наглядно конструировать экранные формы и управляющие элементы, а также запись всей программы или ее частей при помощи средства MacroRecorder (автоматическая запись макроса). Довольно часто при решении простейших задач достаточно средств визуального программирования. Но если требуется создать более сложное приложение с содержательной обработкой данных, то на первый план выступает сам язык программирования VBA.

2 ПРОЕКТ VBA

VBA относится к языкам объектно-ориентированного программирования. Чтобы начать писать программы на VBA, необходимо знать элементы языка, инструкции, их синтаксис и правила составления кода, объекты и их иерархию. Код VBA хранится в проектах. Проекты содержат модули, а модули включают процедуры и функции. Процедуры и функции содержат описания и инструкции на языке VBA.

Каждое приложение MS Office хранит проекты по-своему:

• Word сохраняет проекты в документах и шаблонах с бинарным форматом (DOC и DOT) и в документах и шаблонах с форматом Open XML с поддержкой макросов (DOCM и DOTM).

• Excel сохраняет проекты в рабочих книгах и шаблонах с бинарным форматом (XLS и XLT) и в рабочих книгах и шаблонах с форматом Open XML с поддержкой макросов (XLSM и XLTM).

• Access сохраняет проекты в файлах базы данных (MDB и ACCDB).

• PowerPoint сохраняет проекты в презентациях и шаблонах с бинарным фор-матом (PPT, PPS и POT) и в презентациях с поддержкой макросов (PPTM, PPSM и POTM).

Для программирования на VBA нет необходимости в обязательном порядке устанавливать какую-либо версию транслятора VB (Visual Basic). Для работы с VBA, например в Microsoft Excel, необходимо отобразить в главном меню команду *Разработчик*. Это можно выполнить последовательностью команд *Файл-Параметры-Настройка ленты* и установить отметку возле опции *Разработчик* в правом списке. После этого можно будет открывать окно редактора VBA, выполняя щелчок по кнопке *Visual Basic* панели инструментов команды *Разработчик* главного меню (клавиатурная комбинация запуска редактора VBA - Alt-F11).

В окне проекта первой строкой записывается оператор объявления процедуры (программы)

Sub имя()

Завершающей строкой процедуры является строка

End Sub

Для запуска программы из редактора используют клавишу F5. Для запуска программы из документа — Alt+F8.

В случае использования команды Debug.Print для вывода результатов в окно интерактивной работы (Immediate) следует отобразить это окно, нажав Ctrl+G.

Рассмотрим элементы, которые могут входить в состав проекта.

Процедуры. Процедура представляет собой поименованную последовательность совместно выполняемых инструкций (операторов). Существуют процедуры типа *Sub, Function и Property*. Имя процедуры всегда определяется на уровне модуля.

Процедура *Sub* (подпрограмма) представляет собой набор команд, с помощью которого можно решить определенную задачу. При запуске процедуры выполняются ее команды, затем управление передается в вызвавшее данную процедуру приложение или процедуру. В процедурах типа Sub обычно содержится большинство исполняемых программ.

Процедура *Function* (функция) отличается от предыдущего типа процедур тем, что обязательно возвращает одно результирующее значение. Функции обычно используют для вычисления численного значения, получения символьной строки в результате операций над текстом или для определения логического значения.

Процедура *Property* (свойство) применяется для ссылки на свойство объекта. В частности, этот тип процедур используется для задания или определения значений пользовательских форм и модулей.

Модули. В состав проекта можно включать модули кода (или просто модули). Модули служат для размещения в них процедур. Число модулей в составе проекта может определять-

ся в зависимости от сложности проекта. Внутрь одного модуля имеет смысл помещать процедуры, обеспечивающие решение одной прикладной задачи.

Макросы представляют собой подпрограммы (процедуры типа *Sub*), не имеющие параметров и размещаемые в модуле. Макросы, создаваемые путем протоколирования действий пользователя, размещаются в модуле New Macros.

Модули класса служат для размещения в них описания объектов, создаваемых в VBA. Каждый модуль класса содержит описание одного типа объекта.

Формы позволяют решить задачу организации ввода и вывода данных. В среде VBA можно разрабатывать формы, а также процедуры обработки событий, возникающих в этих формах.

3 ТИПЫ ДАННЫХ И ПЕРЕМЕННЫЕ

3.1 Типы данных в VBA

Любая программа выполняет обработку данных, которые размещаются в памяти, используя переменные или константы. Переменная представляет собой поименованный участок оперативной памяти, содержимое которого может изменяться при работе с приложением. Константа определяет неизменяемые данные программы.

Как переменная, так и константа характеризуются следующими элементами: именем, типом, областью видимости – областью, где имя переменной видимо и, значит, возможен доступ к ее значению. При задании имен переменных нужно соблюдать следующие правила:

- имена должны начинаться с буквы;
- имя не должно содержать пробел, точку, восклицательный знак и символы @, &, $, #;
- имя должно включать не более 255 символов;
- не рекомендуется задавать имена, совпадающие с ключевыми словами VBA;
- имя каждой переменной должно быть уникальным.

Определение типа данных задает:

- область возможных значений;
- структуру организации данных;
- операции, определенные над данными этого типа.

В VBA поддерживается определенная классификация типов данных.

Простые и сложные типы данных. Простые типы данных (табл. 3.1) могут принимать единственное значение. Сложные типы данных могут принимать множества значений данных в текущий момент работы программы.

Встроенные и определенные пользователем типы данных. Встроенные типы данных изначально принадлежат языку программирования и составляют его базис. На основе встроенных типов программист может строить собственные, им определенные типы данных, следуя определенным правилам.

Статические и динамические типы данных. Для данных статического типа память отводится в момент объявления – требуемый размер данных известен в момент объявления. Для данных динамического типа размер данных в момент объявления не известен, и память им выделяется в процессе выполнения программы.

Таблица 3.1 – Простые типы данных языка VBA

Имя типа	**Возможные значения**	**Требуемая память**
Boolean	True, False	2 байта
Byte	0..255	1 байт
Integer	-32768…+32767	2 байта
Long	Примерно -2 000 000 000 …+ 2 000 000 000	4 байта
Decimal	Примерно 30 десятичных цифр; можно указать число цифр после десятичной запятой	12 байтов
Single	-3,4E38…-1,4E-45 для отрицательных значений; 1,4E–45…3,4E38 для положительных значений	4 байта
Double	1,7E308…-4,9E-324 для отрицательных значений; 4,9E-324…1,7E308 для положительных значений	8 байтов
Currency	Десятичные числа с фиксированной позицией запятой. Возможны 15 цифр до запятой и 4 после	8 байтов
String	Есть два вида строк:	10 байтов +1

Имя типа	Возможные значения	Требуемая память
	• строки фиксированной длины имеют до 2^{16} символов; • строки переменной длины имеют до 2^{31} символов	байт на символ в обычной кодировке и 2 байта в кодировке Unicode
Date	Даты изменяются в диапазоне от 1 января 100 г. до 31 декабря 9999 года	8 байтов
Object	Ссылка на объект (указатель)	4 байта
Variant	Универсальный тип, значением которого могут быть данные любого из перечисленных выше типов, объекты, значения NULL и значения ошибок ERROR.	Зависит от контекста, но не менее 16 байтов

3.2 Объявление простых переменных

Объявление переменных можно осуществлять на двух уровнях – уровне процедуры и уровне модуля. Для объявления переменных используют операторы Dim, Public, Private и Static. Оператор Dim можно использовать на обоих уровнях, Public, Private – на уровне модуля, Static – только на уровне процедуры.

Переменные, объявленные на уровне процедуры, называются локальными по отношению к ней, то есть они видны только в данной процедуре. Переменные уровня модуля являются глобальными. Область видимости глобальных переменных может распространяться:

- на все процедуры модуля, в котором они объявлены. Такие глобальные переменные, называемые закрытыми (Private), должны быть объявлены на уровне модуля: либо оператором Private, либо оператором Dim;
- на весь программный проект – все процедуры всех модулей данного проекта. Такие глобальные переменные, называемые открытыми (Public), должны быть объявлены оператором Public.

Локальные переменные уровня процедуры могут быть объявлены оператором Static, что делает их статическими. Статические переменные характеризуются тем, что при выходе из процедуры их значения сохраняются, в отличие от обычных локальных переменных. Просто данные значения становятся временно недоступными до тех пор, пока процедура не будет вызвана вновь. При использовании статических переменных необходима их первоначальная инициализация. VBA инициализирует переменные при их объявлении следующими значениями:

- 0 – для числовых значений;
- пустая строка ("") – для строк переменной длины;
- строка, содержащая нули, – для строк фиксированной длины;
- Empty (значение, указывающее на отсутствие инициализации) – для типа Variant;
- для массивов и записей (типа, определенного пользователем) каждый элемент инициализируется в соответствии с указанными правилами.

Объявление простых переменных имеет следующий синтаксис:

{Dim | Public | Private | Static} <имя переменной> [As <имя типа>]
[,<имя переменной> [As <имя типа>]]

Каждое объявление переменной связывает имя переменной с ее типом, заданным конструкцией As. Если после имени переменной отсутствует указанная конструкция, то переменной будет присвоен тип Variant. Отметим также, что переменные типа Decimal объявляются с использованием функции CDec.

Приведем пример описания переменных:

```
Public Const pi As Double = 3.1415926 ' Глобальная переменная

Public Sub MY_Prnt()
' Локальные переменные
  Dim      B As Byte,      I As Integer,   L As Long
  Dim      Sng As Single, D As Double,  C As Currency
  Dim      Sf As String,   Sv As String,   Dat As Date
  Dim      O As Object,   V
  Debug.Print "Sng=", Sng, "D=", D, "C=", C
  Debug.Print "B=", B, "I=", I, "L=", L
  Debug.Print "Sf=", Sf, "Dat=", Dat, "Sv=", Sv
  If O Is Nothing Then Debug.Print "Объект не определен"
  If V = Empty Then Debug.Print "Variant переменные не инициализированы"
End Sub
```

Результат отладочной печати после запуска на исполнение данной процедуры выглядит следующим образом:

```
Sng = 0          D = 0          C = 0
B   = 0          I  = 0         L = 0
Sf  =          Dat = 0:00:00      Sv=
Объект не определен.
Variant переменные не инициализированы.
```

3.3 Объявление констант

Синтаксис объявления констант:

[Public | Private] Const <имя константы> [As type] = <константное выражение>

Пример объявления константы:

```
Public Const pi As Double = 3.1415926
```

Как и переменные, именованные константы можно объявлять на уровне процедуры или модуля. В первом случае используется только ключевое слово Const, во втором – дополнительно можно задать спецификаторы Public или Private. По умолчанию глобальные константы имеют статус Private.

3.4 Массивы

Массив – это самый распространенный структурный тип данных. Массив представляет собой упорядоченную совокупность данных одного типа. Порядок элементов задается индексами, то есть порядковыми номерами элементов в соответствующей структуре. В VBA массивы могут одномерными и многомерными. Допустимое число измерений около 60.

При описании массивов к уже знакомой структуре описания переменных добавляется необходимость указания размерности:

{ Dim | Public | Private | Static} <имя переменной> (<список размерностей>) [As <имя типа>]

Каждое измерение в списке отделяется запятой и определяется заданием верхней и нижней границы изменения индексов. Массивы – это структуры с прямым доступом к элементам. Доступ осуществляется посредством указания имени массива и индекса элемента. Индекс элемента прописывается в круглых скобках.

Приведем пример:

```
Public Sub MyArray()
    Const LowBound As Integer = –5, HighBound As Integer = 5
    Dim MyArr(LowBound To HighBound) As Byte
    Dim I As Integer
    Debug.Print "Элементы массива MyArr:"
    For I = LowBound To HighBound
        MyArr(I) = I + 6
        Debug.Print MyArr(I)
    Next I
End Sub
```

При исполнении данной процедуры будут напечатаны числа от 1 до 11.

Рассмотренный пример показывает работу со статическим массивом. Количество элементов такого массива определено в момент объявления его в программе. VBA имеет мощное средство работы с массивами – динамические массивы. Массив считается динамическим, если при первоначальном объявлении не указывается его размерность, но в последующем она может быть определена и переопределена оператором ReDim. Размерность определяется динамически в той процедуре и в тот момент, когда она становится фактически известной. При этом границы изменения индексов можно задавать не только как константы, но и как выражения, зависимые от переменных. Если же при переопределении массива задать ключевое слово Preserve, можно сохранить все ранее полученные значения элементов и расширить массив, добавив новые элементы.

Приведем пример работы с динамическим массивом:

```
'Объявление динамического массива на уровне модуля
Public Vector() As Integer

Public Sub DinArray()
'Определяется фактическая размерность массива Vector
    Dim N As Byte, I As Byte
    N = InputBox("Введите число элементов вектора")
    ReDim Vector(1 To N)
    For I = 1 To N
        Vector(I) = 2 * I + 1
    Next I
' Массив расширяется с сохранением ранее вычисленных элементов
    ReDim Preserve Vector(1 To 2 * N + 1)
    For I = N + 1 To 2 * N + 1
        Vector(I) = 2 * I
    Next I
'А теперь печать
    Debug.Print "Элементы Vector:" & Chr(13)
    For I = 1 To N * 2 + 1
        Debug.Print Vector(I)
```

```
    Next I
End Sub
```

При исполнении данной процедуры будет выдан запрос на определение количества элементов массива, затем печать выведенных значений. Например, если ввести число 10, то вначале будут напечатаны нечетные числа от 3 до 21, а затем четные от 22 до 42.

4 ОПЕРАЦИИ И ВСТРОЕННЫЕ ФУНКЦИИ

4.1 Выражение. Правила построения выражений в VBA

В основе программирования на любом алгоритмическом языке лежит построение выражений. Выражение строится из переменных, констант, встроенных функций с использованием знаков, операций и скобок. Запись выражения задает правило вычисления его значения и типа.

Перечислим основные операции, которые можно использовать в VBA при построении выражений (табл. 4.1).

Таблица 4.1 – Операции и их приоритет

Приорите	Арифметическ ие	Сравнен ия	Логические	Описание некоторых операций
1	Возведение в степень (^)	Равенство (=)	Отрицание (Not)	При возведении в степень основание и показатель могут быть арифметическими выражениями любого типа. Результат имеет тип Double
2	Унарный минус (–)	Неравенство (<>)	Конъюнкция (And)	
3	Умножение, деление (*, /)	Меньше (<)	Дизъюнкция (Or)	
4	Деление нацело (\)	Больше (>)	Исключительное ИЛИ (Xor)	Деление нацело определено над целочисленными данными (применимо и к вещественным данным) и дает результат целого типа. Исключительное ИЛИ требует, чтобы один из операндов имел значение, отличное от True
5	Остаток от деления нацело (mod)	Меньше или равно (<=)	Эквивалентность (Eqv)	Операция mod определена над данными целого типа и возвращает результат целого типа
6	Сложение, вычитание (+, –)	Больше или равно (>=)	Импликация (Imp)	Среди логических операций определена операция следования (импликация), ложная в единственном случае – когда посылка истинна, а заключение ложно
7	Конкатенация строк (&)	Подобия (Like), равенство ссылок (Is)		Операция Like проверяет соответствие строки образцу. Операция Is, определенная над объектами, не проверяет равенство самих объектов, она проверяет совпадение ссылок. Ссылки должны задавать один и тот же объект

Окончание табл. 4.1

При построении выражений необходимо учитывать следующее:

- если выражение содержит операции разных категорий, то первыми выполняются арифметические операции, затем операции сравнения, и последними – логические;
- все операции сравнения имеют один и тот же приоритет. Арифметические и логические операции выполняются в соответствии с указанным приоритетом;
- одна и та же операция, записанная несколько раз подряд, или операции одного приоритета выполняются слева направо – из двух операций первой выполняется та, которая стоит левее в записи выражения;
- скобки позволяют изменить указанный порядок вычисления выражения, поскольку выражения в скобках имеют наивысший приоритет и вычисляются первыми. Внутри скобок действует обычный порядок следования;
- операция конкатенации не является арифметической, но при ее появлении в выражении она выполняется после всех арифметических операций, но до вычисления операций сравнения.

Следует отметить, что операторы Not, And, Or и Xor используются также для работы с числами в двоичном формате.

4.2 Функции обработки числовых данных

Помимо вышеуказанных операций VBA предоставляет набор математических функций, расширяющий возможности обработки числовых данных. Перечислим базовые функции:

- **Abs(число)** – абсолютное значение числа;
- **Atn(число)** – арктангенс (в радианах) аргумента, задающего тангенс угла;
- **Cos(число)** – косинус угла. Аргумент число задает угол в радианах;
- **Sin(число)** – синус угла;
- **Exp(число)** – экспонента, т.е. результат возведения в степень числа *e*;
- **Log(число)** – натуральный логарифм числа;
- **Rnd[(число)]** – результат представляет число, равномерно распределенное случайное число в интервале [0–1]. Если аргумент число не задан или больше нуля, то порождается очередное случайное число. Если он равен 0, то результатом будет предыдущее случайное число, а если число меньше нуля, то каждый раз порождается одно и то же число, определяемое аргументом. Перед тем, как получить последовательность случайных чисел, необходимо вызвать функцию Randomize для инициализации последовательности;
- **Sng(число)** – знак числа (если число больше нуля – 1, равно нулю – 0, меньше нуля – –1);
- **Sqr(число)** – квадратный корень;
- **Tan(число)** – тангенс угла.

4.3 Обработка строковых данных

Операции над строками

Над строковыми переменными, определенными в проектах VBA, допустимо выполнение двух видов операций: сравнения и конкатенации строк.

Операция конкатенации используется для сцепления двух или нескольких строк. Обозначается данная операция знаком «+» либо знаком «&». В случае, если применяется первый знак, то в качестве аргументов выражения, определяющего операцию конкатенации, должны выступать переменные или константы строкового типа. В случае применения второго знака (&) – один из аргументов может быть переменной или константой типа число или дата. Например:

```
Dim stroka As String
Dim Ver As Single
    Ver = 1
    stroka = "Компьютерная  " + "подготовка " + " часть " & Ver
    MsgBox (stroka)
```

При сравнении строк применимы обычные операции сравнения. При этом сравнение может быть осуществлено в соответствии с расположением строк в словаре либо побитно. Второй тип сравнения обладает чувствительностью к регистру. Чтобы определить тип сравнения, необходимо в начале модуля поместить инструкцию Option Compare Text | Binary.

Сравнение строк с образцом осуществляется с использованием операции Like. При задании образца используются специальные символы (табл. 4.2), позволяющие разнообразить операцию сравнения.

Таблица 4.2 – Специальные символы, используемые при задании шаблона

Символы	Интерпретация	Примеры
*	Любой текст – произвольное число	Шаблону Agent* соответствуют все тексты, начинающиеся со слова Agent. Строки Agent007 и Agent Майор Пронин удовлетворяют шаблону
?	Один любой символ	Шаблону К?к удовлетворяют строки Кок и Кук
#	Любая цифра от 0 до 9	Шаблону Agent### соответствует 1000 различных строк, среди которых и Agent007
[множество символов]	Любой символ, принадлежащий множеству	Задать множество можно с помощью перечисления и интервалов. Шаблону К[аоу]к удовлетворяют слова «Как», «Кок», «Кук»
[!множество символов]	Любой не принадлежащий множеству символ	Шаблону [!а–я] удовлетворяет символ, не являющийся буквой русского алфавита

Основные функции обработки строковых переменных

Функция **Len(string)** возвращает длину строки (число символов), которая задана аргументом String.

Функция **InStr** определяет позицию первого вхождения одной строки внутри другой строки. Синтаксис:

InStr([start,]string1, string2[, compare])

Необязательный аргумент start задает позицию, с которой начинается поиск (по умолчанию – с первого символа строки). String1 – строка, в которой осуществляется поиск, string2 – подстрока, вхождение которой ищется. Необязательный аргумент compare указывает способ сравнения строк. Его значение по умолчанию 0 используется для выполнения двоичного сравнения; 1 задает посимвольное сравнение без учета регистра.

Функция **Left(string, length)** выделяет в строке string указанное число length символов слева.

Функция **Rigth(string, length)** выделяет в строке string указанное число length символов справа.

Функция **Mid(string, start[, length])** позволяет выделить из строки string подстроку длины length, начиная с позиции start.

Функции **Ltrim(string), Rtrim(string), Trim(string)** возвращают копию строки, из которой удалены пробелы, находящиеся в начале строки, в конце строки или в начале и в конце строки соответственно.

Функции **LCase(string), UCase(string)** возвращают копию строки, символы которой приведены к нижнему или к верхнему регистру.

4.4 Работа с датами и временем

Для того чтобы обеспечить программисту возможность корректно работать с датами и временем, VBA предоставляет специальный тип Date, хранящий дату и время. Представление дат занимает 4 байта памяти. Целая часть хранит число дней от начальной даты, а дробная часть хранит время от полуночи. Начальной датой является 30 декабря 1899 года. При работе с данными значениями чаще всего используются специальные встроенные функции.

Присваивание значений

При присваивании значений типа Date следует заключать дату в специальные ограничители "#" или задавать ее как строковую константу. При задании даты в ограничителях, например #9, May, 99#, она автоматически преобразуется в стандартную форму #5/9/99#.

Пример работы с датами:

```
Public Sub WorkWithDates()
Dim dat1 As Date, dat2 As Date, dat3 As Date

   dat1 = 12
   dat2 = 9 / 5 / 99
   dat3 = #9/5/99#
   Debug.Print dat1, dat2, dat3
     dat1 = "15/7/99"
     dat2 = #5/9/99#
     dat3 = dat3 + 100
     Debug.Print dat1, dat2, dat3
   If dat3 > dat2 Then
     Debug.Print dat3 – dat2
   Else
     Debug.Print dat2 – dat3
   End If
End Sub
```

Результаты выполнения этой программы:

```
11.01.1900      0:26:11      05.09.1999
15.07.1999      09.05.1999   14.12.1999
 219
```

В первом операторе данной программы к начальной дате прибавляется 12 дней, отсюда получается 11 января 1900 года. Второй оператор вычисляется как обычное арифметическое выражение, так как выражение не заключено в ограничители. Полученное дробное число воспринимается как время от начала суток. Далее программой выполняются операции над датами – прибавление целого числа дней, сравнение дат и вычитание дат.

Встроенные функции работы над датами

Date возвращает текущую дату.

Time возвращает текущее время по часам компьютера.

Now возращает значение типа *Variant(Date)*, содержащее текущую дату и время по системному календарю.

DateAdd добавляет и вычитает указанный временной интервал из значения даты:

DatAdd (interval, number, data)

Аргумент *interval* – строка, указывающая тип добавляемого временного интервала, *number* – число временных интервалов, на которые надо изменить дату, *data* – дата, к которой добавляется указанный временной интервал. Допустимые значения аргумента *interval* приведены в таблице 4.3.

DateDiff – определяет время, прошедшее между двумя датами:

DateDiff (interval, date1, date2[, firstdayofweek[, firstweekofyear]])

Таблица 4.3 – Возможные временные интервалы

Значение	Описание
yyyy	Год
Q	Квартал
m	Месяц
Y	День года
D	День месяца
w	День недели
ww	Неделя
H	Часы
N	Минуты
S	Секунды

Параметры имеют следующий смысл:

- *Interval задает тип временного интервала при вычислении разности между датами date1 и date2 – его возможные значения те же, что и для функции DateAdd;*
- *date1* и *date2* – две даты, разность между которыми следует вычислить;
- *firstdayofweek* – константа, указывающая первый день недели (по умолчанию считается, что неделя начинается с воскресенья);
- *firstweekofyear* – константа, указывающая первую неделю года (по умолчанию первой неделей считается неделя, содержащая первое января).

DateSerial – вычисляет значение даты типа Variant(Date) по ее компонентам – году, месяцу, дню:

DateSerial(year, month, day)

Значение каждого аргумента должно лежать в соответствующем диапазоне: 100 – 9999 для года, 1–31 для дней, 1–12 для месяцев. Можно использовать для аргументов числовые выражения для описания относительной даты.

Примеры:
DateSerial(1977–25, 10–2, 18–1)
Результат: 17.08.1952

DateSerial(Year(Now) – 3, Month(Now) – 2, Day(Now) – 1)
Результат на текущий день, например: 24.11.2000

DateValue переводит аргумент-строку в дату.
DateValue(date)

Примеры тестирования данной функции в окне отладки:

?DateValue("25.04.1997")
25.04.1997
?DateValue("04.25.1997")
25.04.1997
?DateValue("25 апреля 1997")
25.04.1997
?DateValue("25–апр–97")
25.04.1997
?DateValue("Апрель, 25, 97")
25.04.1997

?DateValue("25/04/1997")
25.04.1997
?DateValue("25.04")
25.04.2004

Day(date), Month(date), Year(date) по аргументу-дате определяют номер дня, месяца и года.

Hour(time), Minute(time), Second(time) по аргументу, который является числовым или строковым выражением и представляет время, возвращает соответственно часы, минуты и секунды в значении времени.

TimeSerial(hour, minute, second) вычисляет результат Variant(Date), содержащий значение времени, соответствующее указанному часу, минуте, секунде.

TimeValue(time) возвращает значение типа Variant(Date), содержащее время. Аргумент time задается строковым выражением, представляющем время от 0:00:00 (12:00:00 AM) до 23:59:59 (11:59:59 PM).

5 ОПЕРАТОРЫ

5.1 Операторы как основа

VBA – операторный язык. Это значит, что его программы (модули) представляют последовательность операторов. Набор операторов VBA весьма обширен и не уступает в этом таким «большим» языкам, как Паскаль и С. Группа декларативных операторов VBA, служащих для описания объектов, с которыми работает программа (типов, переменных, констант, объектов приложений), уже рассмотрена. Операторы второй группы обеспечивают присвоение и изменение значений этих объектов; операторы третьей группы управляют ходом вычислений, четвертой – работой с каталогами и файлами и т.д. Часть операторов досталась VBA в наследство от предыдущих версий Бейсика и без них можно обойтись при написании новых программ.

При записи текста программ для упрощения чтения, отладки и модификации программы удобней каждый оператор располагать в отдельной строке текста. Следуйте правилу: «Один оператор – одна строка». Но на этой строке разрешается размещать и несколько операторов. В отличие от общепринятого символа разделения операторов «точка с запятой», в VBA символом разделения двух операторов в одной строке служит двоеточие. Заметьте, некоторые операторы, например оператор If, могут стоять лишь на первом месте в строке. И по этой причине каждый оператор, как правило, следует начинать с первой строки. Лишь иногда разумно группу операторов присваивания размещать в одной строке. Чаще возникает другая ситуация – оператор слишком длинный, и его текст не виден полностью на экране дисплея, что затрудняет чтение и понимание программы. В этом случае оператор следует продолжить в одной или нескольких строках. Чтобы продолжить (перенести) оператор на следующую строку, используется пара символов пробел-подчеркивание « _». Например,

```
MyAddress = "дом: " & Number & " улица: " & Street _
    & " город: " & City
```

Перед оператором в строке может стоять метка – последовательность символов, начинающаяся с буквы и кончающаяся двоеточием «:». Метки можно размещать в отдельных строках перед теми операторами, которые они должны помечать. Они нужны для операторов перехода типа *GoTo*, использование которых считается «дурным тоном» в структурном программировании. Но иногда без меток и переходов обойтись трудно – в частности, при указании входов в обработчике ошибок в некоторых процедурах.

5.2 Оператор комментария

Комментарии на исполнение программы не влияют, но необходимы как признак «хорошего стиля». Офисные программы используются многократно и не раз модернизируются в процессе жизни. Вы можете сэкономить на комментариях и написать, а затем и отладить небольшой модуль без них. Но уже через неделю никто, в том числе и Вы, не сможет понять его действие и модифицировать нужным образом. Затраченные на это усилия и время намного превзойдут «экономию». Любая строка текста программы может заканчиваться комментарием. Комментарий в VBA начинается апострофом (') и включает любой текст, расположенный в строке правее. Обычно в комментариях описываются задачи, решаемые модулями; функции, выполняемые процедурами; смысл основных переменных (если он неясен из имен); алгоритмы работы процедур. Полезно также комментировать операторы вызова внешних данных для данного модуля процедур, объясняя их действия.

Другое применение комментарии находят при отладке программ. Если требуется исключить из программы некоторые операторы (например, вызовы еще не реализованных или сомнительных процедур), то достаточно перед ними поместить апостроф, например, при выполнении последовательности операторов:

$x = x + z;$

$'z = fun(x, z);$

$y = y * z.$

Функция fun для вычисления нового значения *z* во второй строке не вызывается и мешает проверить правильность значения y.

В VBA имеется еще один способ выделения комментариев с помощью ключевого слова *Rem*. Такой комментарий (в отличие от комментария, начинающегося апострофом) должен отделяться в строке от предыдущего оператора двоеточием:

```
weight=weight+z : Rem Увеличение веса
value=weight*price        'Новая стоимость
```

При отладке VBA-программ часто приходится временно комментировать отдельные участки текста, а иногда и целые страницы. В этом случае закомментировать или снять комментарии довольно утомительно, и поэтому разумно пользоваться двумя полезными командами меню Edit: Comment Block и UnComment Block. Они позволят автоматически закомментировать или снять комментарий с выделенного блока.

5.3 Присваивание

Операторы присваивания – основное средство изменения состояния программы (значений переменных и свойств объектов). Они уже многократно использовались в примерах в предыдущих разделах. В VBA есть несколько видов операторов присваивания: *Let, Lset, Rset* и *Set*.

Оператор Let

С помощью этого оператора происходит «обычное» присвоение значения выражения переменной или свойству.

Его синтаксис:

[Let] переменная = выражение

Ключевое слово *Let*, как правило, опускается. ***Переменная*** является именем переменной или свойства; ***выражение*** задает значение, присваиваемое переменной. Его тип должен соответствовать типу переменной. Нарушение этого условия, например, при попытке присвоить числовой переменной строковое значение, приводит к тому, что при компиляции появляется сообщение об ошибке. Переменным типа *Variant* можно присваивать значения разных типов, например, строковых или числовых выражений. Строковой переменной можно присваивать любое значение типа *Variant*, кроме *Null*. Числовой же переменной значение типа *Variant* можно присвоить, только если оно может быть преобразовано к числу.

Оператор *Let* можно применять для присваивания одной переменной типа запись ***значения*** другой такой переменной. Заметьте, это возможно, только если обе они – переменные – одного определенного пользователем типа.

Примеры:

```
Public Sub Assign1()
    Dim MyStr As String, MyInt As Integer
    Let MyStr = "Здравствуйте, зайчик! "  'С ключевым словом Let
    MyInt = 5              'Без него. Обычный вариант.
    Debug.Print MyStr, MyInt
End Sub
```

Оператор Lset

Этот оператор служит для присваивания строк с одновременным выравниванием слева, а также для копирования записи одного определенного пользователем типа в запись другого типа.

Его синтаксис:

Lset СтрПеременная = СтрВыражение
Lset переменная1 = переменная2

Здесь ключевое слово *Lset* обязательно, *СтрПеременная* – имя строковой переменной, *СтрВыражение* – выражение строкового типа. Во втором варианте *переменная1* – имя переменной некоторого определенного пользователем типа, в которую выполняется копирование, *переменная2* – имя переменной, возможно, другого пользовательского типа, значение которой копируется.

Результатом присваивания строк всегда является строка той же длины, что и у *СтрПеременная*. Если при этом *СтрВыражение* короче, добавляются пробелы справа; длинее – лишние символы справа удаляются.

При втором варианте оператора двоичное присваивание записи из участка памяти, отведенного *переменной2*, копируется в участок памяти, отведенный *переменной1*, – при этом данные не преобразуются в соответствии с типами элементов (полей) записи, и если типы соответствующих элементов обеих записей не совпадают, результат операции трудно предсказать (часто выдается сообщение о несоответствии типов). Поэтому этот вариант следует использовать в том случае, когда типы всех соответствующих элементов записей совпадают и имеют одинаковый размер.

Примеры:

```
Public Sub Assign2()
   Dim Str1 As String, Str2t As String

    Str1 = "0123456789"          ' Начальное значение
    Str2 = "abcd"
    Debug.Print Str1, Str2

    Lset Str2 = Str1       ' Результат – "0123"
    Lset Str1 = "Влево" ' ' Результат – "Влево"
    Debug.Print Str1, Str2

End Sub
```

Результаты отладочной печати:

```
0123456789      abcd
Влево      0123
```

В следующем примере происходит корректное копирование одной записи в другую; типы соответствующих элементов в определенных пользователем типах *MyType1, MyType2* совпадают:

```
Type MyType1
    age As Integer
    cost As Long
End Type

Type MyType2
```

```
    year As Integer
    pay As Long
End Type

Public Sub Assign3()
    Dim my1 As MyType1
    Dim my2 As MyType2

    my1.age = 50
    my1.cost = 45666
    my2.year = 1997
    my2.pay = 22000
    Debug.Print my1.age, my1.cost, my2.pay, my2.year

    Lset my2 = my1
' после этого присвоения my2.year = 50 и my2.pay = 45666
    Debug.Print my1.age, my1.cost, my2.pay, my2.year

End Sub
```

Результаты отладочной печати:

```
50          45666  22000  1997
50          45666  45666  50
```

Если изменить тип элемента *pay* и *MyType2* на *Integer*, то будет напечатано значение *my2.pay*, равное *0*.

Оператор Rset

Этот оператор присваивает значение строковой переменной с выравниванием справа:

Rset СтрПеременная = СтрВыражение

СтрПеременная – имя строковой переменной, *СтрВыражение* – выражение строкового типа. В отличие от оператора *Lset*, оператор *Rset* нельзя использовать для копирования переменных записей. Результатом присвоения строк всегда является строка той же длины, что и *СтрПеременная*. Если при этом *СтрВыражение* короче, добавляются пробелы слева, длинее – лишние символы слева удаляются.

Примеры:

```
Public Sub Assign4()
    Dim Str1, Str2, Str3

    Str1 = "0123456789 "  ' Начальное значение
    Str2 = "abcd "
    Debug. Print Str1, Str2

    Rset Str2 = Str1       ' Результат – "0123 "
    Rset Str1 = "Вправо "        ' Результат – "Вправо "
    Rset Str3 = Str1       ' Результат – пустая строка " "
    Debug.Print Str1, Str2, Str3
End Sub
```

Результаты отладочной печати:

0123456789 abcd
Вправо 0123

5.4 Управляющие операторы

Набор управляющих операторов VBA делает честь любому хорошо структурированному языку программирования. Циклы с возможной проверкой условия в начале, в конце и в середине работы оператора, обычный оператор *If* и оператор разбора случаев *Case* – все эти средства позволяют организовать процесс вычислений надежно и эффективно в соответствии с лучшими традициями программирования.

Условный оператор If Then Else End If

Этот общепринятый в языках программирования оператор управления вычислениями позволяет выбирать и выполнять действия в зависимости от истинности некоторого условия.

Имеется два варианта синтаксиса: в одну строку и в форме блока. В первом случае он имеет вид:

If условие Then [операторы1] [Else операторы2]

Во втором случае оператор расположен на нескольких строках:

If условие Then
[операторы]
[ElseIf условие-n Then
[операторы-n]…
[Else
[ИначеОператоры]]
End If

Здесь *условие* обязательно в обоих вариантах. Оно может быть числовым или строковым выражением со значениями *True* или *False* (*Null* трактуется как *False*). *Операторы1* и *операторы2* – это последовательности из одного или нескольких разделенных двоеточием операторов. По крайней мере, одна из этих последовательностей должна быть недоступной. Если *условие* истинно (*True*), выполняется последовательность *операторы1*, ложно (*False*) – *операторы2*. Форма условного оператора выделяется по наличию в строке, начинающейся *If условие Then*, текста после Then, отличного от комментария. Если такой текст есть, считается, что использована форма в одну строку, нет – оператор должен иметь форму блока. В этом случае подблоки вида:

[ElseIf условие-n Then
[операторы-n]…

могут отсутствовать, либо повторяться несколько раз;

подблок:

[Else
[ИначеОператоры]]

также необязателен, а закрывающийся оператор *End If* необходим. По крайней мере, одна из последовательностей *операторы, операторы-n…* или *ИначеОператоры* должна быть непустой. Если *условие* истинно, выполняются *операторы*, нет – отыскивается первое истинное *условие-n* и выполняются *операторы-n*. Если все эти условия ложны, выполняются *ИначеОператоры*. После выполнения одной (возможно, пустой) последовательности операторов управление передается оператору, следующему за *End If*.

Примеры:

```
Public Sub MinMax2(ByVal X As Integer, ByVal Y As Integer, _
                   Min As Integer, Max As Integer)
    'Оператор If  в виде блока
    If X > Y Then
        Max = X
        Min = Y
    Else
        Max = Y
        Min = X
    End If
End Sub

Public Sub If1()
    Dim Max As Integer, Min As Integer
    Call MinMax1 (2+3, 2*3, Min, Max)
    Debug.Print Max, Min
    Call MinMax2 (2+3, 2*3, Min, Max)
    Debug.Print Max, Min
End Sub
```

В этом примере в процедуре MinMax определяется максимальное из двух значений переменных. Приведем еще один пример, где показана работа с записями. Вначале определим тип данных:

```
Private Type Man
    Age As Byte
    Name As String
End Type
```

А теперь приведем процедуру, в которой будем работать с переменной типа *Man*.

```
Public Sub BoyOrMan()
 Dim Person As Man
 Dim SomeBody As Byte
 Person.Age=InputBox("Введите число от 10 до 20", "Возраст", 15)

 If Person.Age  > 15 And Person.Age  <= 20 Then
     SomeBody = 1

 ElseIf Person.Age <= 15 Then SomeBody = 2

 End If

 If SomeBody=2 Then
    Debug.Print "Это мальчик! "
 ElseIf  SomeBody=1 Then
    Debug.Print "Это мужчина! "
 Else
    Debug.Print "Не знаю, кто это "
 End If
End Sub
```

Результаты ее работы зависят от того, какой возраст Вы введете в окне ввода.

Оператор выбора Select Case

Этот оператор производит разбор случаев и в зависимости от значения анализируемого выражения выбирает и исполняет одну из последовательностей операторов.

Его синтаксис:

Select Case Выражение-текст
[Case Список выражений-n
[операторы-n]]
[Case Else
[ИначеОператоры]]
End Select

Выражение-текст должно присутствовать обязательно. Оно может быть произвольным выражением с числовым или строковым значением. *СписокВыражений-n* должен присутствовать в строке, начинающейся ключевым словом *Case* (Случай). Выражения в этом списке отделяются запятыми и могут иметь одну из форм:

- выражение;
- выражение-нижняя-граница *To* выражение-верхняя-граница;
- *If* оператор-сравнение – выражение.

Первая форма задает отдельные значения, вторая и третья – позволяет задавать сразу целые диапазоны (области) значений. Последовательность операторов *операторы-n* необязательна. Она будет исполнена, если соответствующий *СписокВыражений-n* является первым списком, сопоставимым с текущим значением *Выражения-текста* (оно явно содержит это значение, либо оно попадает в один из заданных в списке диапазонов). После исполнения операторов последовательности *операторы-n* проверка на соответствие другим спискам выражений не производится, и управление передается на оператор, следующий за *End Select*. Необязательная последовательность *ЕслиОператоры* выполняется, если ни один из списков выражений несопоставим со значением *Выражения-текста*.

Пример:

```
Public Sub Case1()
Dim Before As Integer
Dim CurrentYear As Integer, Str As String
'Инициализация переменных:
    CurrentYear = 1999
    Before = InputBox("Сколько лет тому назад? ", "Когда", 10)
    Select Case CurrentYear - Before
      Case 1954 To 1969, 1971 To 1974, 1982, Is < 1970
        Str = "Годы учебы"
      Case 1972 To 1989
        Str = "Годы воспитания"
      Case Else
        Str = "Прочие годы"
    End select
    Debug.Print Str
End Sub
```

Здесь, если перед выполнением выбора *Before = 20*, значением текстового выражения будет 1979, и будет работать второй вариант (*"Годы воспитания "*). При *Before = 25* значение 1974 сопоставимо с двумя списками, но для исполнения будет выбран лишь первый вариант (*"Годы учебы "*).

Диапазоны значений можно задавать и для строк. При этом их значения считаются упорядоченными лексикографически. Например, возможен такой список выражений:

```
Case "everything ", "nuts" To "soup"
```

Задаваемое им множество строк включает строку *"everything"* и все строки от *"nuts"* до *"soup"* (например, *"onion"*).

Цикл For…Next

Цикл For…Next позволяет повторять группу операторов заданное число раз.
Его синтаксис:

```
For счетчик_цикла = начало To конец [Step шаг]
    тело цикла
Next [счетчик_цикла]
```

Здесь *счетчик_цикла* – это числовая переменная. В начале выполнения цикла она принимает значение, задаваемое числовым выражением *начало*. Переменная *счетчик_цикла* не может иметь тип Boolean или быть элементом массива. Числовое выражение *конец* задает заключительное значение счетчика цикла. Оно вычисляется до начала исполнения тела цикла и не меняется, даже если входящие в него переменные изменяют в теле цикла свои значения. Числовое выражение *шаг* необязательно. Его значение также вычисляется в начале цикла и прибавляется к счетчику цикла всякий раз, когда завершается выполнение тела цикла и вычисление достигает строки *Next* [*счетчик_цикла*]. Если шаг цикла явно не указан, по умолчанию он равен 1. *Тело цикла* – это последовательность операторов, которая будет выполнена заданное число раз. Значение переменной *счетчик_цикла*, при котором происходит завершение цикла, зависит от знака параметра *шаг*. Если шаг положителен, цикл завершится, когда впервые выполнится условие:

счетчик_цикла > конец

Если *шаг* цикла отрицателен, условие его завершения:

счетчик_цикла < конец

Это условие проверяется перед началом работы цикла, а затем – после каждого прибавления *шага* к счетчику цикла в операторе *Next*. Если оно выполнено, управление передается на оператор, следующий за *Nex*t, нет – выполняются операторы из тела цикла. Завершить цикл *For…Next* можно и с помощью оператора *Exit For*. Такие операторы могут быть расположены в тех местах тела цикла, где требуется из него выйти, не дожидаясь выполнения условия завершения.

Примеры.
В данном примере три вложенных цикла *For…Next* использованы для вычисления двух целочисленных матриц, инициализированных случайными числами. Затем результирующая матрица проверяется на наличие в ней нулевого значения:

```
Public Sub For1()
    Dim A(1 To 5, 1 To 5) As Integer
    Dim B(1 To 5, 1 To 5) As Integer
    Dim C(1 To 5, 1 To 5) As Integer
    Dim I As Integer, J As Integer, K As Integer
    Dim Res As String
  'Инициализация матриц A и B случайными числами в интервале
  [-10,+10]
    VBA.Randomize
    For I = 1 To 5
      For J = 1 To 5
```

```
    ' Получение случайного числа Rnd и преобразование в целое
      A(I, J) = Int(21 * Rnd) – 10
    Next J
  Next I
  For I = 1 To 5
    For J = 1 To 5
      B(I, J) = Int(21 *Rnd) - 10
    Next J
  Next I

  ' Вычисление произведения матриц
  For I = 1 To 5
    For J = 1 To 5
      C(I, J) = 0
      For K = 1 To 5
          C(I, J) = C(I, J) + A(I, K) * B(K, J)
      Next K
    Next J
  Next I

  Res = "No"
  C(2, 2) = 0
  ' Проверка на нулевое значение
  For I = 1 To 5
    For J = 1 To 5
      If C(I, J) = 0 Then
          Debug.Print "Индексы: ", I, J
          Res = "Yes"
          Exit For
      End If
    Next J
  Next I
  Debug.Print Res
End Sub
```

Обратите внимание, что оператор выхода *Exit For* прекращает выполнение только внутреннего цикла, т.е. проверка на ноль будет осуществляться в каждой строке матрицы, независимо от существования нулей в предыдущих строках. Сделаем еще несколько замечаний по поводу оператора *For Next*:

- по окончании цикла счетчик цикла сохраняет свое значение в момент выхода, и его можно использовать, например, для анализа преждевременного выхода из цикла. Заметим, что в большинстве языков программирования значение счетчика цикла считается неопределенным по завершении цикла;
- в операторе *Next* можно не указывать имя переменной, задающей счетчик, – это имя подразумевается по умолчанию, как завершающее последний открытый оператор *For*. Однако и эту возможность не следует использовать. Рекомендуется всегда явно указывать имя счетчика в операторе *Next*;
- допускается менять значение счетчика в теле цикла, но делать этого не следует никогда. К сожалению, система не следит за этим и допускает подобные безобразия. В этом случае цикл никогда не завершится:

```
Public Sub For2()
Dim A(1 To 5) As Integer
```

```
Dim i As Integer
    For i = 1 To 5 To 1 Step -1
        A(i) = i
        i = i + 1
    Next i
End Sub
```

Обычно попытка изменить значение счетчика цикла в его теле означает, что вместо цикла *For…Next* следовало бы применить цикл другого вида, например *Do…Loop*.

Цикл Do…Loop

В этом случае повторяется блок операторов, пока заданное условие является истинным или пока оно не станет истинным.

Имеется четыре варианта синтаксиса этого цикла. В двух первых вариантах условие проверяется в начале цикла:

```
Do [{While | Until} условие]
тело цикла
Loop
```

В других двух вариантах условие проверяется в конце цикла:

```
Do
тело цикла
Loop [{While | Until} условие]
```

Здесь *условие* является числовым или строковым выражением со значениями *True* или *False*. Вообще, оно необязательно. Значение *Null* условия трактуется как *False*. *Тело цикла* – это последовательность операторов, которая будет выполняться, пока *условие* остается истинным, если перед ним стоит ключевое слово *While*, или пока оно остается ложным – в вариантах цикла с ключевым словом *Until*. Таким образом, циклы вида *While условие* эквивалентны циклам вида *Until Not условие*. Кроме того, в *тело цикла* может входить оператор *Exit Do*, выполнение которого сразу прекращает цикл и передает управление оператору, непосредственно следующему за *Loop*. В случае нескольких вложенных циклов *Do…Loop* оператор *Exit Do* завершает лишь самый внутренний цикл, в теле которого он расположен.

Примеры.

В данном примере реализованы три варианта поиска по образцу с проверкой условия в начале цикла, в конце цикла и в середине цикла для варианта поиска по образцу с барьером:

```
Public Sub Loop1()
Const Size = 5
Dim X() As Integer
Dim I As Integer
Dim Found As Boolean
Const pat = 7
'Инициализация случайными числами в интервале [1, 11]
    ReDim X(1 To Size)
    Randomize
    For i = 1 To Size
      X(i) = Int(11 * Rnd)
    Next i

' Поиск по образцу с проверкой в начале цикла
    i = 1: Found = False
```

```
    Do While (i <= Size) And (Not Found)
      If X(i) = pat Then
           Found = True
      Else: i = i+1
      End If
    Loop
    If Found Then
      Debug.Print "Найден образец!"
    Else: Debug.Print "Образец не найден!"
    End If

' Поиск по образцу с проверкой в конце цикла
    i = 1: Found = False
    Do
      If X(i) = pat Then
        Found = True
      Else: i = i+1
      End If
    Loop Until Found Or (i = Size + 1)
    If Found Then
      Debug.Print "Найден образец!"
    Else: Debug.Print "Образец не найден!"
    End If

' Поиск с барьером
    ReDim Preserve X(1 To Size + 1)
    X(Size + 1) = pat
    i = 1
    Do
      If X(i) = pat Then Exit Do
      i = i + 1
    Loop
    If i = size + 1 Then
      Debug.Print " Образец не найден!"
    Else: Debug.Print "Образец найден!"
    End If
End Sub
```

Цикл While…Wend

Этот цикл повторяет выполнение последовательности операторов, пока заданное условие не станет ложным.

Его синтаксис:

```
While условие
    тело цикла
Wend
```

Здесь условие и тело цикла такие же, как и для цикла *Do…Loop*. Только для этого вида цикла не предусмотрен оператор выхода *Exit*. Фактически, цикл *While…Wend* – частный случай цикла *Do…Loop* – оставлен в языке для совместимости с предыдущими версиями.

Цикл For Each…Next

Цикл For Each…Next повторяет заданную последовательность операторов для каждого элемента массива или набора.

Его синтаксис:

For Each элемент In группа
тело цикла
Next [элемент]

Здесь *элемент* – переменная, которая пробегает в качестве значений элемента массива. В случае цикла по массиву *элемент* обязан быть переменной типа *Variant*. *Группа* – это имя массива, для элементов которого выполняется цикл. Цикл не применим для массивов, тип элементов которых определен пользователем, так как такие элементы не могут быть значениями переменной типа *Variant*. В таких массивах можно использовать цикл вида *For…Next*. *Тело цикла* – последовательность операторов, выполняемая для каждого элемента набора или массива, – может содержать операторы *Exit For*, позволяющие прервать выполнение цикла и передать управление оператору, следующему за *Next* (обычно такой выход происходит при выполнении некоторого условия, проверяемого в операторе *If…Then…Else*). Указывать переменную *элемент* после ключевого слова *Next* не обязательно, но желательно.

Примеры.

В примере создается динамический массив, размерность которого определяется в ходе решения задачи. Каждый набор чисел, записываемый в массив, сортируется. На последнем этапе массив распечатывается. В цикле типа *For…Each* формируется строка вывода, элементы которой выбираются из массива *Stroka*.

```
Private Mas() As Integer
Public Sub ForEach1()
Dim i As Integer
Dim s As String
 ReDim Mas(10) As Integer
 Make_Mas 10
 Call Print_Mass(10, "Начальное ", "значение ", "массива X(10)")
 Sort_Mas 10
 Call Print_Mass(10, "Отсортированный ", "массив X(10)")
 ReDim Mas(20) As Integer
 Make_Mas 20
 Call Print_Mass(20, "Начальное  ", "значение  ", "массива X(20)")
 Sort_Mas 20
 Call Print_Mass(20, "Отсортированный ", "массив X(20)")

End Sub

Sub Print_Mass(ByVal n As Integer, ParamArray Stroka())
Dim i As Integer
Dim s As String
Dim sl
 For Each sl In Stroka
   s = s & sl
 Next

 Debug.Print (s)
 For i = 0 To n - 1
   Debug.Print Mas(i);
 Next i
```

```
  Debug.Print
End Sub

Sub Make_Mas(n)
  Dim i As Integer
  For i = 0 To n - 1
    Mas(i) = Rnd * 20 - 2
  Next i
End Sub
Sub Sort_Mas(ByVal n)
  Dim i, j, k As Integer
  Dim Min As Integer
  For i = 0 To n - 1
    Min = Mas(i): k = i
    For j = i + 1 To n - 1
      If Mas(j) < Min Then
        k = j
        Min = Mas(j)
      End If
    Next j
    Mas(k) = Mas(i)
    Mas(i) = Min
  Next i

End Sub
```

Кроме рассмотренных управляющих операторов, VBA содержит доставшиеся в наследство от прежних версий операторы перехода по метке *GoTo*, перехода по метке с возвратом *GoSub…Return* и условные операторы перехода по меткам *On…GoSub* и *On…GoTo*.

6 ПРОЦЕДУРЫ И ФУНКЦИИ

6.1 Классификация процедур

Процедуры и функции являются минимальными модульными конструкциями, из которых строится проект.

Процедура (функция) – это программная единица VBA, включающая операторы описания ее локальных данных и исполняемые операторы. Обычно в процедуру объединяют регулярно выполняемую последовательность действий, решающую отдельную задачу или подзадачу. Особенность процедур VBA состоит в том, что они работают в мощном окружении Office и могут использовать в качестве элементарных действий большое количество встроенных методов и функций, оперирующих с разнообразными объектами этой системы. Поэтому структура управления типичной процедурой прикладной офисной системы проста: она состоит из последовательности вызовов встроенных процедур и функций. Последовательность эта управляется небольшим количеством условных операторов и циклов.

Процедуры VBA можно классифицировать по нескольким признакам:

- по способу использования (вызова) в программе;
- по способу запуска процедуры на исполнение;
- по способу создания кода процедуры;
- по месту нахождения процедуры в проекте.

Процедуры VBA подразделяются на подпрограммы и функции. Первые описываются ключевым словом *Sub*, вторые – *Function*.

По способу создания кода процедуры делятся на обычные, разрабатываемые вручную, и на процедуры, код которых создается автоматически генератором макросов; их называют макро-процедурами или командными процедурами, поскольку их код – это последовательность вызова команд соответствующего приложения Office. Любую макропроцедуру можно изменить вручную, дописав в нее требуемую последовательность операторов.

По способу запуска процедур на исполнение можно выделить в отдельную группу процедуры, запускаемые автоматически при наступлении того или иного события, – так называемые процедуры обработки событий.

По положению в проекте различают процедуры, находящиеся в специальных программных единицах – стандартных модулях, модулях классов и модулях, связанных с объектами, реагирующими на события.

6.2 Синтаксис процедур и функций

Описание процедуры Sub в VBA имеет следующий вид:

[Private | Public | Static] Sub имя ([список-аргументов])
тело-процедуры
End Sub

Ключевое слово **Public** в заголовке процедуры используется, чтобы объявить процедуру общедоступной, т.е. разрешить вызывать ее из всех других процедур всех модулей всех проектов. Если модуль, в котором описана процедура, содержит закрывающий оператор **Option Private**, то процедура доступна лишь модулям своего проекта. Альтернативный ключ **Private** используется, чтобы закрыть процедуру от всех модулей, кроме того, в котором она описана. По умолчанию процедура считается общедоступной.

Ключевое слово **Static** означает, что значение локальных переменных будут сохраняться в промежутках между вызовами процедуры (используемые процедурой глобальные переменные при этом не сохраняются).

Параметр ***имя*** **–** это имя процедуры, удовлетворяющее стандартным условиям VBA на имена переменных.

Необязательный параметр ***список-аргументов*** – это последовательность разделенных запятыми переменных, задающих при вызове процедуры передаваемые параметры. Аргументы, или формальные параметры (с их типами), задаваемые при описании процедуры, всегда представляют только имена. В то же время при вызове процедуры ее аргументы – фактические параметры – могут быть не только переменными, но и выражениями.

Приведем пример процедур со списком параметров. Предположим, что необходимо вычислить сумму двух различных произведений, отличающихся по виду сомножителей, которые являются элементами некоторых массивов.

$$S = \prod_{k=1}^{4} f_k + \prod Sin(f_k), \quad \text{где} \quad f = \{5; \ -3; \ 1; \ 2\}.$$

Для решения данной задачи создадим две процедуры: процедуру *Main* с нулевым списком параметров и процедуру *Product*, которая имеет три параметра.

```
Private Sub Main()
Dim f(4), f1(4), s, p2, p1 As Single
Dim i As Integer
   For i = 1 To 4
     f(i) = Val(InputBox("Введите f"))
     f1(i) = Sin(f(i))
   Next
   Product 4, f, p1
   Call Product(3, f1, p2)
   s = p1 + p2
   Debug.Print "Сумма произведений равна  "; s
End Sub
Sub Product(ByVal k As Integer, ByVal z, p)
   Dim i As Integer
   p = 1
   For i = 1 To k
     p = p * z(i)
   Next
End Sub
```

Здесь после ввода в память значений элементов массивов *f* и *f1* дважды вызывается процедура *Product*. Обратите внимание на различный синтаксис вызова процедуры. При использовании ключевого слова *Call* список параметров должен быть заключен в скобки. Ключевое слово *ByVal* определяет список входных параметров, в данном примере это параметры *k, z*. Параметр *p* в данном примере является выходным, следовательно, вычисленное значение произведения передается через этот параметр в точку вызова процедуры.

В отличие от процедуры, функция не имеет выходных параметров и не вызывается, подобно процедуре, как самостоятельная инструкция, а включается в состав выражения, значение которого может быть присвоено какой-либо программной переменной. Описание функции имеет следующий вид:

```
[Private | Public | Static] Function имя ([список-аргументов]) _
                                        As типЗначения
   тело-функции
```

```
End Function
```

Рассмотрим в качестве примера определение функции *SQ*, которая возвращает значение площади треугольника, если заданы декартовы координаты трех вершин. Очевидно, что у данной функции 6 аргументов – 3 пары координат вершин.

```
Function SQ(x1 As Single, y1 As Single, x2 As Single, y2 As Single, _
            x3 As Single, y3 As Single) As Double
    SQ = Abs((x1 * (y3 - y2) + x2 * (y1 - y3) + x3 * (y2 - y1)) / 2)
    SQ = CInt(SQ * 100) / 100
End Function


Private Sub Main()
Dim s As Double
    s= SQ(2.7, 9.1, 14.1, 16.4, 17.9, 2)
    Debug.Print "Площадь треугольника равна "; s
End Sub
```

Обратите внимание, что в заголовке функции указаны не только тип аргументов, но тип значения самой функции. Другой особенностью описания функции является то, что результат работы операторов, стоящих в теле функции, присваивается имени функции: *SQ = CInt(SQ * 100) / 100.*

7 ИНТЕГРИРОВАННАЯ СРЕДА VBA

7.1 Окно редактирования проекта

Написание и отладка программ осуществляется с использованием интегрированной среды Visual Basic. Среда (в дальнейшем мы будем называть ее редактором) VBA активизируется командой **Сервис/Макрос/Редактор Visual Basic** непосредственно с рабочего окна программы Word либо Excel. Вид окна редактора Visual Basic показан на рисунке 7.1.

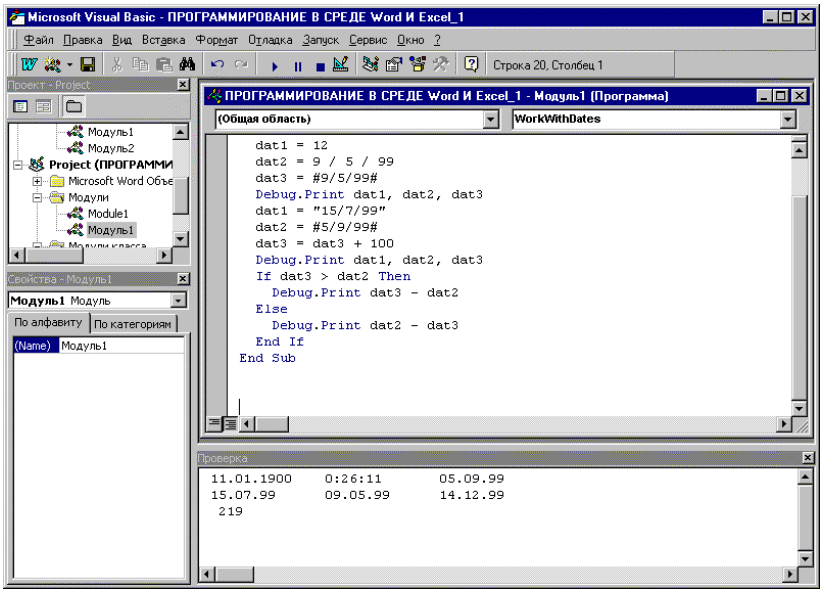

Рис. 7.1 – Окно VBA

Интерфейс редактора Visual Basic является стандартным для всех приложений Windows. В верхней части окна находится строка меню, ниже располагается панель инструментов **Стандартная**. В рабочем поле могут отображаться следующие окна: **Проект, Свойства, Проверка, Локальные переменные, Контрольные значения, Просмотр объектов, Программа.** Вызов всех перечисленных окон выполняется с помощью почти одноименных меню **Вид.**

Программное окно или окно модуля служит для написания любых программ (макросов, процедур любого типа) VBA. Под строкой заголовка окна содержится два списка: список объектов модуля и список процедур, связанных с выбранным объектом. В окне модуля можно задать отображение отдельной процедуры или модуля в целом. Переключение режимов отображения выполняется с помощью кнопок, расположенных в левом нижнем углу окна модуля. Тексты программ вводятся непосредственно в окне модуля.

Чтобы добавить в проект VBA новый модуль, необходимо выполнить следующие команды:

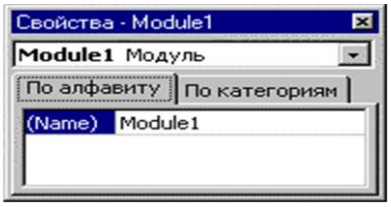

Рис. 7.2 – Окно Свойства

1) выбрать в окне проекта проект, в состав которого добавляется модуль;

2) выполнить команду **Модуль** меню **Вставка**;

3) присвоить модулю имя. Для этого необходимо открыть окно **Свойства** (рис. 7.2) и в диалоговом окне ввести новое имя.

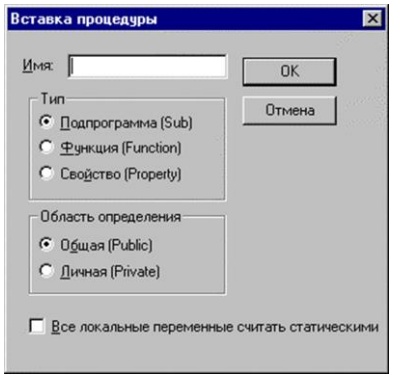

Рис.7.3 – Окно Вставка процедуры

Формирование и отладка процедур

Для добавления процедуры необходимо выполнить следующие команды:

1) открыть окно модуля;

2) выполнить команду **Процедура** меню **Вставка** (рис. 7.3);

3) в предложенном окне запроса выбрать тип процедуры (**Sub**, **Function** или **Property**);

4) в поле **Имя** ввести имя процедуры: любая последовательность символов;

5) задать область определения функции (Public – общая или Private – личная);

6) при необходимости установить переключатель **Все локальные переменные считать статическими;**

7) нажать кнопку ОК.

В результате в окне модуля выводится пустая процедура с заданным именем. Для окончательного создания процедуры нужно ввести ее тело, то есть последовательность операторов и выполнить отладку.

Отладка процедур проводится для выявления и исправления синтаксических и алгоритмических ошибок.

Синтаксические ошибки заключаются в неправильной записи инструкций VBA, неверном указании имен процедур и переменных. Такие ошибки выявляются автоматически на этапе компиляции.

Алгоритмические ошибки проявляются в том, что процедура не дает ожидаемых результатов. Такие ошибки выявляются путем пошагового выполнения процедуры и контроля промежуточных результатов с помощью средств отладки VBA.

Пошаговое выполнение процедур можно задать с помощью панели инструментов **Отладка** или с помощью команд одноименного меню.

Меню **Отладка** предоставляет четыре варианта пошагового выполнения процедуры:

1. Команда **Шаг с заходом** предписывает выполнить одну строку процедуры и перейти к следующей. В случае, если очередная строка содержит вызов процедуры, то осуществляется переход к ее первой инструкции.

2. Команда **Шаг с обходом** предписывает выполнить одну строку процедуры и перейти к следующей. Вызываемая процедура выполняется как одна строка. Этот вариант применяется в том случае, когда вызываемая процедура уже отлажена.

3. Команда **Шаг с выходом** задает выполнение текущей процедуры до конца и переход к следующей инструкции в вызывающей процедуре.

4. Команда **Выполнить до текущей** позиции задает выполнение процедуры от текущей инструкции до инструкции, в которой был предварительно установлен курсор.

Отладка процедур сопровождается переходом в режим прерывания. Переход в этот режим происходит в следующих случаях:

- при прерывании выполнения макроса с помощью комбинации <Ctrl>+<Break>;
- при пошаговом выполнении процедуры;
- при достижении инструкции Stop;
- при достижении точек останова.

Для выхода из режима прерывания нужно в среде редактора Visual Basic выполнить команду **Сброс** меню **Запуск.**