Технология программирования

Блок 3. Сложные структуры данных и методы их обработки

Тема 2. Пользовательские процедуры и функции

Цели изучения темы:

- сформировать у студентов навыки работы с пользовательскими процедурами и функциями.

Задачи изучения темы:

- ознакомить студентов с понятием пользовательских процедур, особенностями их применения и функционирования.

В результате изучения данной темы Вы будете
знать:

- основные принципы использования пользовательских процедур и функций;
- классификацию пользовательских процедур;
- особенности функций и аргументов и специфику их использования;

уметь:

- программировать задачи с использованием различных процедур;
- анализировать и применять соответствующие параметры и модификаторы при написании программ;

владеть:

- набором необходимых понятий и инструментов для использования пользовательских процедур.

Учебные вопросы темы:

1. Пользовательские процедуры.
2. Классификация процедур.
3. Прерывание процедур и функций VBA.
4. Некоторые особенности функций и аргументов

Основные термины и понятия, которые Вам предстоит изучить:

процедура, подпрограмма, параметр, функция, аргумент, модификатор, рекурсивная процедура.

Вопрос 1. Пользовательские процедуры

При разработке макросов часто требуется выполнять одни и те же законченные действия в различных его частях. Чтобы избежать многократного набора кода для этих действий, целесообразно описать их в виде подпрограммы и обращаться к ней по мере необходимости.

Если вам придется создавать реальные программы, вы очень скоро убедитесь в том, что некоторые операции (например, запись каких-либо значений, создание и сохранение документов и т.д.) могут быть нужными в различных местах программы и занимать порой несколько десятков строчек кода. Нерационально каждый раз копировать эти участки кода в нужное место. *Во-первых* - увеличивается размер текста программы, *во-вторых* - если вам понадобится изменить что-нибудь в часто используемом наборе операторов - придется искать все такие участки и редактировать каждый из них. Гораздо удобнее будет выделить часто используемый участок в виде самостоятельной процедуры и каждый раз, когда он будет нужен, вызывать его с помощью одной лишь строчки кода.

Ранее мы уже пользовались встроенными процедурами и функциями VBA. Может показаться, что их достаточно много для решения любой задачи. Но даже при таком разнообразии порой нужны совершенно особые процедуры, которые пользователь может создать самостоятельно.

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 в том, что они работают в мощном окружении Office 97 и могут использовать в качестве элементарных действий большое количество встроенных методов и функций, оперирующих с разнообразными объектами этой системы. Поэтому структура управления типичной процедуры прикладной офисной системы довольно проста: она состоит из последовательности вызовов встроенных процедур и функций, управляемой небольшим количеством условных операторов и циклов. Ее размеры не должны превышать нескольких десятков строк. Если в Вашей процедуре несколько сотен строк, это, скорее всего, значит, что задачу, решаемую процедурой, можно разбить на несколько самостоятельных подзадач, для решения каждой из которых следует написать отдельную процедуру. Это облегчит понимание программы и ее отладку. Разумеется, эти замечания носят неформальный методологический характер, поскольку сам VBA никак не ограничивает ни размер процедуры, ни сложность ее структуры управления.

Различают подпрограммы-процедуры и подпрограммы-функции. Описываются подпрограммы при помощи инструкций ***Sub...End Sub***—

это подпрограмма-процедура, инструкций ***Function...End Function***— это подпрограмма-функция. Подпрограмма ***Function*** и подпрограмма ***Sub*** являются самостоятельной программой, которая может получать аргументы, выполнять последовательность операторов и изменять значения своих аргументов.

Главное отличие процедуры от функции заключается в том, что функция возвращает в точку вызова некое значение, которое, как правило, является результатом обработки переданной функции информации. А процедура лишь выполняет какие-либо действия, но ничего в точку вызова не возвращает.

Например, очевидно, что при обработке такой последовательности команд: **num_A = num_B + val(str_C)** вместо выражения **val(str_C)** подставляется числовое значение переменной **str_C**, найденное благодаря функции **Val**. Или, скажем, такое выражение: **num_A = InputBox("Введите число")**. Здесь функция **InputBox** возвращает введенное пользователем число в переменную **num_A**.

Процедуры, как уже было сказано, лишь выполняют какие-то действия, не возвращая никаких значений в точку вызова. Например, если мы вызываем окно сообщения с надписью, не настраивая никаких дополнительных параметров, то мы можем решить, что имеем дело с процедурой. Такой вызов: **MsgBox ("Привет"),** будет отлично работать. В то же время, **MsgBox** может вести себя как функция, возвращая в точку вызова код нажатой кнопки при настройках кнопок окна сообщения отличных от стандартных.

Для работы подпрограммам иногда необходимо передать данные, которые называются *параметрами* или *аргументами* подпрограммы. Параметры передаются в подпрограммы при ее вызове при помощи перечисления их после имени подпрограммы. При этом описание действий в подпрограмме осуществляется с использованием формальных параметров. Обращение к подпрограмме осуществляется с фактическими параметрами, которые должны соответствовать формальным по числу, типу и месту расположения.

Вопрос 2. Классификация процедур

Процедуры VBA можно классифицировать по нескольким признакам: по способу *использования (вызова)* в программе, по способу *запуска процедуры на выполнение,* по способу *создания кода процедуры,* по *месту нахождения кода* процедуры в проекте.

Процедуры VBA подразделяются на подпрограммы и функции. Первые описываются ключевым словом Sub, вторые - Function. Для VBA характерно использовать термин *подпрограмма*. Однако, вместо него в программировании более распространен термин *процедура*. Иногда, это может приводить к недоразумениям, поскольку в зависимости от контекста под процедурой понимается как подпрограмма, так и

функция. Различие между этими видами процедур скорее синтаксическое, так как преобразовать процедуру одного вида в эквивалентную процедуру другого вида совсем не сложно. В языке C/C++, как известно, есть только функции и нет процедур, по крайней мере формально.

По способу создания кода процедуры делятся на ***обычные, разрабатываемые "вручную***", и на процедуры, ***код которых создается автоматически генератором макросов (MacroRecoder)***; их называют также ***макро-процедурами*** или командными процедурами, поскольку их код – это последовательность вызовов команд соответствующего приложения Office 97. И это разделение в известной степени условно, так как довольно типичны процедуры, каркасы которых, созданные генератором макросов, затем изменяют и дописывают вручную.

По способу запуска процедур на выполнение можно выделить в отдельную группу процедуры, ***запускаемые автоматически*** при наступлении того или иного события, - мы называем их процедурами обработки событий.

По положению в проекте различаются процедуры, находящиеся в специальных программных единицах - ***стандартных модулях***, модулях ***классов*** и модулях, ***связанными с объектами***, ***реагирующими на события***.

Еще один специальный тип процедур - процедуры-свойства Property Let, Property Set и Property Get. Они служат для задания и получения значений закрытых свойств класса.

Главное назначение процедур во всех языках программирования состоит в том, что при их вызове они изменяют состояние программного проекта, - изменяют значения переменных (свойства объектов), описанных в модулях проекта. У процедур VBA сфера действия шире. Их главное назначение состоит в изменении состояния системы документов, частью которого является изменение состояния самого программного проекта. Поэтому процедуры VBA оперируют, в основном, с объектами MS Office. Заметьте, есть два способа, с помощью которых процедура получает и передает информацию, изменяя тем самым состояние системы документов. Первый и основной способ состоит в использовании параметров процедуры. При вызове процедуры ее аргументы, соответствующие входным параметрам получают значение, так процедура получает информацию от внешней среды, в результате работы процедуры формируются значения выходных параметров, переданных ей по ссылке, тем самым изменяется состояние проекта и документов. Второй способ состоит в использовании процедурой глобальных переменных и объектов, как для получения, так и для передачи информации.

Процедуры

Оператор ***Sub...End Sub*** Описывает имя, аргументы и текст программы, составляющий тело процедуры Sub.

Sub *имя [(списока аргументов)]*

Операторы

End Sub

имя — обязательный элемент, это имя процедуры Sub, удовлетворяющее =: стандартным правилам именования переменных.

список аргументов — необязательный элемент. Список переменных, представляющий параметры, которые передаются в процедуру Sub при ее вызове (формальные параметры). Имена переменных разделяются запятой.

операторы— любая группа операторов, выполняемых в процедуре Sub, находятся между Sub и End Sub.

Список аргументов имеет вид:

[Optional] [ByVal | ByRef] [ParamArray] *имя параметра*[( )] [As type] [= defaultvalue]

Если параметр является необязательным, необходимо также как часть объявления указать его значение по умолчанию. Синтаксис для указания значения по умолчанию выглядит следующим образом:

Optional [ByVal | ByRef] *имя параметра* As *тип данных* = *значение по умолчанию*

Аргументы в VBA могут быть переданы процедуре двумя способами:

- **ByVal** – передача аргумента по значению. Это значит, что процедуре передаётся только значение (то есть, копия аргумента), и, следовательно, любые изменения, сделанные с аргументом внутри процедуры, будут потеряны при выходе из неё.
- **ByRef** – передача аргумента по ссылке. То есть процедуре передаётся фактический адрес размещения аргумента в памяти. Любые изменения, сделанные с аргументом внутри процедуры, будут сохранены при выходе из процедуры.

Не допускается определение процедуры ***Sub*** внутри другой процедуры ***Sub*** или ***Function***.

Оператор вызова подпрограмм Call

Обращение к процедуре осуществляется с помощью оператора Call.

Call *имя подпрограммы* [*параметры*]

Синтаксис инструкции Call содержит следующие элементы:

имя подпрограммы — обязательный элемент, содержит имя вызываемой про- цедуры.

параметры — необязательный элемент, это разделяемый запятыми список переменных, массивов или выражений, передаваемых в процедуру (фактические параметры).

Для передачи в процедуру полного массива следует воспользоваться именем массива с пустыми скобками.

Функции

Оператор ***Function...End Function***

Описывает имя, аргументы и текст программы, составляющий тело подпрограммы- функции Function.

Function ***имя [(список аргументов)]*** **[As** ***тип*****]**

Операторы

[*имя = выражение*]

End Function

имя — обязательный элемент. Содержит имя подпрограммы-функции Function, удовлетворяющее стандартным правилам именования переменных;

список аргументов — необязательный элемент, это список переменных, представляющий параметры, которые передаются в подпрограмму Function при ее вызове (формальные параметры). Имена переменных разделяются запятой.

тип — необязательный элемент. Тип данных значения, возвращаемого подпрограммой Function.

операторы— элемент, содержащий любую группу операторов, выполняемых внутри процедуры Function.

выражение — возвращаемое значение подпрограммой Function.

Использование подпрограммы **Function** (нестандартной функции) аналогично использованию стандартных функций. Обращение к ней можно записать , например, в правой части оператора присваивания, при этом указывается имя подпрограммы- функции и в круглых скобках – фактические параметры т.е. *список аргументов*, заменяющих формальные параметры в операторе **Function**

Пример. Опросить пятерых пользователей, обработав и записав их ответы. Самое главное в этом примере не то, куда мы будем записывать ответы, и как их будем обрабатывать, а то, что блок кода будет повторяться для каждого пользователя. Сначала напишем этот блок в

расчете на одного пользователя. Для этого добавим в форму Microsoft Word кнопку UserData и создадим для нее такой обработчик события

```
Private Sub UserData_Click()
    Dim str_Name As String
    Dim byte_Age As Byte
    MsgBox ("Здравствуйте, вы пользователь № 1")
    str_Name = InputBox("Введите ваше имя")
    byte_Age = InputBox("Введите ваш возраст")
    'Здесь находится блок обработки и
    'записи введенных данных
End Sub
```

Давайте подумаем, что нужно сделать с кодом при переходе ко второму пользователю. Очевидно, что мы сможем работать с теми же переменными, что и в первом случае, будем использовать те же операторы. Единственным, что изменится, будет надпись в первом окне сообщения. Вместо "Здравствуйте, вы пользователь №1" будет выведено "Здравствуйте, вы пользователь №2".

Это значит, что мы можем использовать весь код, который находится в данный момент в обработчике, для создания собственной процедуры.

```
Public Sub UserInput(UserNumber As Integer)
      'пользовательская процедура
      'для ввода и обработки данных

End Sub
```

Внутри процедуры – там, где будет располагаться ее тело, мы разместили комментарии.

UserInput - это имя процедуры. Мы выбираем его самостоятельно. В скобках после имени находится объявление переменной, которую можно передать процедуре в качестве параметра. В нашем случае это переменная, имеющая имя UserNumber и тип Integer. Вспомните, как мы работаем с функцией MsgBox - вызывая ее мы пишем такой код: MsgBox ("Привет"). В данном случае мы передаем функции MsgBox текстовый параметр "Привет", который она выведет на экран. Точно так же, мы сможем передать нашей процедуре числовой параметр типа Integer, который будет "виден" ей под именем UserNumber. Вызов нашей процедуры для первого пользователя из обработчика события Click будет выглядеть так:

```
UserInput (1)
```

Тот номер, который мы указали в скобках, будет присвоен переменной UserNumber и мы сможем пользоваться ей внутри процедуры.

Ключевое слово ***Sub*** означает, что перед нами процедура. А модификатор доступа ***Public*** означает, что эту процедуру можно вызвать из любого модуля проекта. Существуют и другие

модификаторы доступа, в частности, ***Private***. Он дает доступ к процедуре только из того модуля, где она объявлена

В общем коде будет изменяться лишь номер пользователя. Поэтому перепишем строку приглашения таким образом, чтобы для каждого номера пользователя она выводила бы собственный текст. После этого текст процедуры будет выглядеть так

```
Public Sub UserInput(UserNumber As Integer)
      'пользовательская процедура
      'для ввода и обработки данных
      Dim str_Name As String
      Dim byte_Age As Byte
      MsgBox ("Здравствуйте, вы пользователь № " +
Str(UserNumber))
      str_Name = InputBox("Введите ваше имя")
      byte_Age = InputBox("Введите ваш возраст")
End Sub
```

Теперь доработаем обработчик события так, чтобы с их помощью мы могли решить поставленную выше задачу. Ниже вы можете видеть код обработчика события **Click** для кнопки

Код с применением процедуры:

```
Private Sub UserData_Click()
      For i% = 1 To 5
            UserInput (i)
      Next i
'Здесь находится блок обработки и
'записи введенных данных
End Sub
```

UserInput (i) вызов процедуры для пользователя с номером **i**

Пример создания и использования **функции.**

Функция возводит переданное ей число во вторую степень.

```
Public Function Square(numOne As Double) As Double
            Square = numOne ^ 2
End Function
```

Объявление функции очень похоже на объявление процедуры. Здесь мы используем модификатор доступности *Public*, дающий доступ к функции из всех модулей. Ключевое слово **Function** означает, что мы объявляем функцию. Следом за ним, в скобках, идет объявление параметров, которые мы можем передать функции. В нашем случае это - одна переменная **num_One** типа Double. После объявления имени и параметров функции следует объявление типа самой функции, или,

точнее - типа возвращаемого функцией значения. В нашем случае с помощью конструкции As Double мы задали тип функции Double. При объявлении функции можно не указывать тип - тогда он автоматически будет установлен в Variant.

Последняя строка объявления функции - **End Function**, означает конец функции. Как и в случае с процедурой, внутри тела функции можно разместить операторы, необходимые для выполнения задачи. *Однако, здесь есть одна очень важная особенность*. После того, как найдено значение, которое функция должна возвратить, в тексте кода функции **нужно присвоить это значение переменной, которая имеет то же имя, что и функция**. В нашем случае, при имени функции **Square** это присвоение выглядит как **Square = num_One ^ 2**.

```
Private Sub UserCalc_Click()
Dim num_Res As Double
Dim num_Input As Double
      num_Input = CDbl(InputBox("Введите число"))
   num_Res = Square(num_Input)
      MsgBox (Str(num_Input) + " во 2-й степени = "
+ Str(num_Res))
End Sub
```

Мы работаем с пользовательской функцией точно так же, как работали бы с одной из встроенных функций. Для наглядности мы использовали переменную ***num_Res***, в которую записываем значение, возвращаемое функцией.

Вопрос 3. Прерывание процедур и функций VBA

Иногда встречаются обстоятельства, при которых нет смысла продолжать выполнение процедуры или функции. VBA имеет операторы ***Exit*** и ***End,*** которые дают возможность либо завершать процедуру или функцию, либо останавливать всю программу.

Для того, чтобы процедура или функция прекратила выполнение, используется одна из двух доступных форм VBA-оператора **Exit**, в зависимости от того, необходимо ли завершить функцию или процедуру:

Завершение выполнения процедуры

Exit Sub

Завершение выполнения функции

Exit Function

Оператор ***Exit Sub*** приводит к тому, что VBA немедленно прекращает выполнение кода процедуры. После выполнения этого

оператора VBA прекращает выполнение текущей процедуры и возвращается к выполнению той процедуры или функции, которая вызвала процедуру, содержащую оператор ***Exit Sub***.

Для полного завершения выполнения программы используется ключевое слово ***End*** в отдельной строке:

Полное завершение выполнения программы

End

При выполнении этого оператора VBA прекращает все выполнение операторов процедуры и функции. Любые имеющиеся переменные перестают существовать и их значения теряются.

Т.к. программа полностью прерывается, необходимо перед оператором ***End*** выводить пользователю сообщение о том, что будет происходить и почему. В противном случае пользователи могут не понять, почему программа, которую они используют, внезапно прекратила работу. Если программа перестанет выполняться вследствие какого-либо действия пользователя, такого как отмена окна ввода, без объяснения, пользователь процедуры может никогда не узнать, почему процедура заканчивается.

Также не следует забывать закрывать рабочие книги или выполнять другие сервисные работы перед выполнением оператора ***End***, чтобы пользователю программы не приходилось доделывать незаконченные операции.

Вопрос 4. Некоторые особенности функций и аргументов

Функции с побочным эффектом

В классическом варианте все аргументы функции являются входными параметрами, и единственный результат вычисления функции - это ее возвращаемое значение., примером является функция **Square** . Чаще всего, однако, используются функции с побочным эффектом, то есть такие функции, которые помимо получения значения функции изменяют значения некоторых результирующих параметров, передаваемых функции по ссылке.

Например,

```
Public Function SideEffect(ByVal X As Integer, ByRef Y As Integer) As Integer
    SideEffect = X + Y
    Y = Y + 1
End Function


Public Sub TestSideEffect()
    Dim X As Integer, Y As Integer, Z As Integer
    X = 3: Y = 5
```

```
    Z = X + Y + SideEffect(X, Y)
    Debug.Print X, Y, Z
    X = 3: Y = 5
    Z = SideEffect(X, Y) + X + Y
    Debug.Print X, Y, Z
End Sub
```

Вот результаты вычислений:

```
3             6             16
3             6             17
```

Как видите, обращаться с функциями, обладающими побочным эффектом, следует с осторожностью. В нашем примере результат вычисления суммы трех слагаемых зависит от порядка их записи. Это противоречит основным принципам математики. Более того, следует понимать, что результат вычисления непредсказуем, поскольку VBA может для увеличения эффективности изменять порядок действий при вычислении арифметического выражения. Поэтому лучше не использовать в выражении вызов функции с побочным эффектом, изменяющей значения входящих в него переменных.

Аргументы, являющиеся массивами

Аргументы процедуры могут быть массивами. Процедуре передается имя массива, а размерность массива определяется встроенными функциями LBound и UBound. Приведем пример процедуры, вычисляющей скалярное произведение векторов:

```
Public Function ScalarProduct(X() As Integer, Y() As Integer) As Integer
    'Вычисляет скалярное произведение двух векторов.
    'Предполагается, что границы массивов совпадают.

    Dim i As Integer, Sum As Integer
    Sum = 0
    For i = LBound(X) To UBound(X)
        Sum = Sum + X(i) * Y(i)
    Next i
    ScalarProduct = Sum
End Function
```

Оба параметра процедуры, передаваемые по ссылке, являются массивами, работа с которыми в теле процедуры не представляет затруднений благодаря тому, что функции **LBound** и **UBound** позволяют установить границы массива по любому измерению. Приведем программу, в которой вызывается функция ***ScalarProduct***:

```
Public Sub TestScalarProduct()
```

```
    Dim A(1 To 5) As Integer, B(1 To 5) As Integer
    Dim C As Variant
    Dim Res As Integer, i As Integer

    C = Array(1, 2, 3, 4, 5)
    For i = 1 To 5
        A(i) = C(i - 1)
    Next i
    C = Array(5, 4, 3, 2, 1)
    For i = 1 To 5
        B(i) = C(i - 1)
    Next i
    Res = ScalarProduct(A, B)
    Debug.Print Res
End Sub
```

Конструкция ParamArray

Иногда, когда в процедуру следует передать только один массив, для этой цели можно использовать конструкцию ParamArray. Следующая процедура PosNeg подсчитывает суммы поступлений Positive и расходов Negative, указанные в массиве Sums:

```
Sub PosNeg(Positive As Integer, Negative As Integer, ParamArray Sums()_ As Variant)
    Dim I As Integer
    Positive = 0: Negative = 0
    For I = 0 To UBound(Sums()) ' цикл по всем элементам массива
        If Sums(I) > 0 Then
            Positive = Positive + Sums(I)
        Else
            Negative = Negative - Sums(I)
        End If
    Next I
End Sub
```

Вызов процедуры PosNeg может иметь такой вид:

```
Public Sub TestPosNeg()
    Dim Incomes As Integer, Expences As Integer

    PosNeg Incomes, Expences, -20, 100, 25, -44, -23, -60, 120
    Debug.Print Incomes, Expences
End Sub
```

В результате переменная Incomes получит значение 245, а переменная Expences - 147. Заметьте, преимуществом использования

массива аргументов ParamArray является возможность непосредственного перечисления элементов массива в момент вызова.

Однако такое использование массива аргументов ParamArray не исчерпывает всех его возможностей. В более сложных ситуациях передаваемые аргументы могут иметь разные типы.

Рекурсивные процедуры

VBA допускает создание рекурсивных процедур, т. е. процедур, при вычислении вызывающих самих себя. Вызовы рекурсивной процедуры могут непосредственно входить в ее тело, или она может вызывать себя через другие процедуры. В последнем случае в модуле есть несколько связанных рекурсивных процедур. Стандартный пример рекурсивной процедуры - функция-факториал Fact(N)= N!. Вот ее определение в VBA:

```
Function Fact(N As Integer) As Long
    If N <= 1 Then          ' базис индукции.
         Fact = 1           ' 0! =1.
    Else                    '  рекурсивный вызов в
случае N > 0.
         Fact = Fact(N - 1) * N
    End If
End Function
```

Однако, в данном случае не рекурсивный вариант вычисления факториала работает почти в три раза быстрее. Кроме проблем со временем исполнения, рекурсивные процедуры могут легко исчерпать и стековую память, в которой размещаются аргументы каждого рекурсивного вызова. Поэтому **избегайте неконтролируемого размножения рекурсивных вызовов** и **заменяйте рекурсивные алгоритмы на итеративные** там, *где использование рекурсии по существу не требуется*.

Польза от рекурсивных процедур в большей мере может проявиться при обработке данных, имеющих рекурсивную структуру (скажем, иерархическую или сетевую). Основные структуры данных (объекты) MS Office вообще-то не являются рекурсивными: один рабочий лист Excel не может быть значением ячейки другого, одна таблица Access - элементом другой и т.д. Но данные, хранящиеся на рабочих листах Excel или в БД Access, сами по себе могут задавать "рекурсивные" отношения, и для их успешной обработки следует пользоваться рекурсивными процедурами.

Если рассмотреть двоичные деревьями поиска. Деревья представляют рекурсивную структуру данных, поэтому и операции над ними естественным образом определяются рекурсивными алгоритмами.

Контрольные вопросы:

1. В чем преимущества использования процедур и функций?
2. Чем процедура отличается от функции?
3. По каким признакам классифицируют функции?
4. Какая команда объявляет процедуру, а какая функцию?
5. Какая команды прерывает исполнение процедуры, а какая прерывает выполнение программы?
6. Каков формат списка аргументов процедуры?
7. Как передать в функцию значение по ссылке?
8. Можно ли передавать в качестве аргумента в функцию массив?
9. Зачем служит конструкция ParamArray?
10. Что такое рекурсивные функции?