Управляющие структуры в Visual Basic

Оператор If … End If

Синтаксис:

```
If Логическое_выражение Then Оператор
```

ИЛИ

```
If Логическое_выражение Then
        Группа_операторов
End If
```

В первом случает, оператор может быть только один. Во втором - сколько угодно.

Условные операторы.

Оператор If … End If

Пример:

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

Скобки здесь не обязательны, но они повышают читабельность кода.

Оператор If...Else...ElseIf...End If

Синтаксис:

```
If Логическое_выражение1 Then
        Группа_операторов
ElseIf Логическое_выражение2 Then
        Группа_операторов
        ...
Else
        Группа_операторов
End If
```

операторы после Else выполняются только в том случае, если ни одно из условий не выполнено

Оператор If…Else…ElseIf…End If

Пример:

```
If (a = b) Then
    b = d
ElseIF (d > c) Then
    a = 20
Else
a=b
End If
```

Оператор Select Case

Синтаксис:

```
Select Case Анализируемое_выражение
Case Значение_1
        Группа_операторов
Case Значение_2
        Группа_операторов
        ...
    Case Значение_N
        Группа_операторов
 Case Else
        Группа_операторов
End Select
```

Анализируемое выражение должно возвращать значение типа, совместимого с типом значений в строка Case

Оператор множественного выбора.

Оператор Select Case

Пример:

```
Select Case iTest
Case 1
    strResult = "iTest = 1"
Case 2, 3, 4
    strResult = "iTest = 2, 3 или 4"
Case 5 To 9
    strResult = "iTest в диапазоне от 5 до 9"
Case iTest < 0
    strResult = "iTest меньше 0"
Case Is > 9
    strResult = "iTest больше 9"
Case Else
    strResult = "iTest равно 0"
End Select
```

Оператор множественного выбора.

Оператор For … Next

Синтаксис:

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

- *Роль счётчика цикла может играть только ранее объявленная переменная целочисленного типа.*
- *По умолчанию значение шага равно 1.*
- *После слова **Next** счётчик можно опустить.*

Оператор For … Next

Пример:

```
Dim c As Integer

Dim iArray(10) As Integer

For c = 0 To 10

   iArray(c) = 5

Next c
```

Оператор For Each … Next

Синтаксис:

```
For Each Имя_Объекта In Имя_Коллекции
        Операции над объектами
Next Имя_Объекта
```

- *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*

Оператор For Each ... Next

Пример:

```
' В этом примере показано, как изменить свойство
' BackColor у всех этикеток, 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
```

Операторы
Do While ... Loop
Do ... Loop While

Синтаксис:

```
Do While Условие_выхода

Группа_операторов
Loop
```

```
Do
    Группа_операторов
Loop While Условие_выхода
```

Отличие заключается в том, что условие выхода проверяется в первом случае перед очередным проходом, а во втором - после выхода.

Операторы цикла.

Операторы
Do While...Loop
Do...Loop While

Пример:

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

Операторы
Do Until ... Loop
Do ... Loop Until

Синтаксис:

```
Do Until Условие_выхода
    Группа_операторов
Loop
```

```
Do
    Группа_операторов
Loop Until Условие_выхода
```

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

Операторы
Do Until ... Loop
Do ... Loop Until

Пример:

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

Операторы 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
n = n-1
Debug.Print  n
If n=5 Then Exit Do
Loop While n > 1
```

Операторы выхода из цикла.

Обработка ошибок в Visual Basic

Оператор On Error GoTo

*Оператор **On Error GoTo** определяет подпрограмму обработки ошибок.*

Синтаксис:

On Error GoTo Метка

- *Если в процессе выполнения программы произошла ошибка, то оператор **On Error GoTo** передаст управление на определенную метку.*
- *При этом стандартный метод обработки ошибок выполнения блокируется.*
- *Подпрограмма обработки ошибок должна завершаться оператором **Resume.***

Оператор On Error GoTo

Пример:

```
'Программа с ошибкой деления
On Error GoTo ErrHandler
    PRINT "Вывод проведенных вычислений"
    PRINT 1000/0 'строка-провокатор
GoTo Met 'здесь основная программа кончается

'начало собственной программы обработки ошибок
ErrHandler:
    PRINT "Найдена ошибка"
    RESUME
Met: End
```

При обнаружении ошибки выполнение основной программы не прекращается и программа продолжает выполняться, в отличие от стандартного метода обработки ошибок.

Обработка ошибок.

Более эффективно обработать ошибку можно, если знать причину ее появления.

- Необходимая информация об ошибке хранится в следующих переменных:

Значение этих переменных можно анализировать в собственной программе обработки ошибок

Обработка ошибок.

Оператор Resume

*Оператор **Resume** можно применять в следующих вариантах:*

Resume	• вызывает повторное выполнение ошибочного фрагмента
Resume Метка	• передает управление на **метку**
Resume Next	• продолжает программу со строки, следующей за предложением, в котором обнаружена ошибка

Обработка ошибок.

Процедуры и функции в Visual Basic

В Visual Basic большинство программ создается из блоков – процедур и функций.

- Весь программный код находится как бы внутри этих процедур.
- Если возникает необходимость в решении какой-либо задачи в любом месте программы, то вызывается процедура

В Visual Basic нельзя ввести код между процедурами.

- Код всегда должен находиться внутри процедуры.

Процедуры и функции.

Процедура - это блок кода, который будет выполняться всякий раз при её вызове.

*Каждая процедура начинается зарезервированным словом **Sub** и заканчивается словом **End**.*

Синтаксис:

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

Процедуры.

Пример:

```
Private Sub ShowMessage(message As String)

    MsgBox message

End Sub
```

Для вызова процедуры достаточно написать её имя:

- **ShowMessage(”Ура!”)**

А можно и так:

- **Call ShowMessage(” Ура!”)**

Скобки, окружаемые параметр обязательны, если перед именем процедуры стоит оператор Call.

- Если Call отсутствует, то скобки ставить не нужно.

Если количество параметров, передаваемых при вызове процедуры, не совпадёт с количеством параметров в объявлении процедуры – Visual Basic сгенерирует ошибку.

Функции.

Функция - это блок кода, который будет возвращать значение.

Этим, и только этим функции отличаются от процедур.

Синтаксис:

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

Функции.

Пример:

```
Public Function Square(number As Long) As Long

    Square = number * number

End Function
```

Вызвать функцию можно так:

- **b = Square (5)**

Можно использовать нашу процедуру для вывода сообщения на экран:

- **ShowMessage Square (5)**

А можно и так:

- **Square 5**

В последнем случае возвращённое функций значение уходит в никуда, но сама функция благополучно выполнится.

arglist имеет следующий вид:

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

- **Optional**
  - Это слово должно предшествовать имени того аргумента, который является необязательным.
  - После необязательного аргумента могут следовать только необязательные же аргументы.
  - Нельзя использовать необязательные аргументы совместно с массивом аргументов (**ParamArray**).

- **ByVal**
  - Аргумент будет передаваться
  - **по значению**
  - Соответствующий аргумент может быть выражением.

- **ByRef**
  - Аргумент будет передаваться
  - **по ссылке**
  - Соответствующий аргумент может быть только переменной.
  - Этот режим передачи аргументов принят по умолчанию

Процедуры и функции.

- **ParamArray**
  - Массив аргументов.
  - Так может быть объявлен только последний аргумент в списке,
  - при этом он будет представлять собой переменную типа **Variant**,
  - содержащую в себе массив.
  - Этим способом можно передавать в процедуру или функцию значительные объемы данных.
  - При использовании массива аргументов нельзя применять свойства
- **Optional**, **ByVal** и **ByRef**.

- **varname**
  - это имя переменной, массива, элемента управления или формы
    - *в последних двух случаях тип принимает значения **Control** и **Form**.*
  - В случае массива после имени массива ставится пара скобок (), внутри которых не указываются границы значений индексов,
    - *что позволяет использовать одну процедуру (функцию) для разного числа элементов массива в каждом конкретном случае.*

Процедуры и функции.

- **defaultvalue**
  - Допустимо только для необязательного аргумента.
  - Задает значение, которое будет автоматически присвоено аргументу в случае его отсутствия.