CHƯƠNG I. Biến và kiểu dữ liệu của VBA

Ngôn ngữ VBA cung cấp một tập hợp đầy đủ các kiểu dữ liệu thông thường, cộng thêm kiểu dữ liệu thông minh gọi là Variant, tự nó thích ứng để lưu giữ bất kỳ kiểu dữ liệu nào.

Các kiểu dữ liệu VB

VB và VBA hỗ trợ các kiểu dữ liệu sau:

Boolean

Có thể chứa các trị logic True or False. Các từ khóa True và False là các hằng định nghĩa sẵn trong VBA.

```
1. var1 = True
2. var2 = False
3. If myBool = False Then
4.    myVar = 4
5.    myBool = True
6. Else
7.    myVar = 5
8.    myBool = False
9. End If
```

Kích thước: 2 byte
Trị: True hoặc False
Trị mặc định: False

Byte

Kiểu phụ về số nhỏ nhất trong VBA.

Kích thước: 1 byte
Trị: 0 đến 255
Trị mặc định: 0

Currency

Dạng số đặc biệt dùng để lưu trữ các giá trị tiền tệ.

Kích thước: 8 byte
Trị: –922,337,203,685,477.5808 đến 922,337,203,685,477.5807
Trị mặc định: 0

Date

Dạng số đặc biệt dùng để biểu diễn ngày và giờ

Kích thước: Eight bytes
Trị: 1 tháng giêng 100 đến 31 tháng mười hai 9999
Trị mặc định: 00:00:00

Thập phân

Kiểu phụ chứa các số thập

Kích thước: 14 bytes
Trị:
- Không có phần thập phân: +/– 79,228,162,514,264,337,593,543,950,335
- Cho tới 28 vị trí thập phân: +/– 7.9228162514264337593543950335
Trị mặc định: 0

Double

Số thực dấu chấm động.

Kích thước: 8 byte
Trị:
- Trị âm: –1.79769313486232E308 to –4.94065645841247E-324
- Trị dương: 1.79769313486232E308 to 4.94065645841247E-324
Trị mặc định: 0

Integer

Số nguyên có dấu. Một bit biểu diễn dấu (âm hoặc dương). Gán trị ngoài vùng giá trị sẽ sinh lỗi runtime.

Kích thước: 2 byte
Trị: –32,768 đến 32,767
Trị mặc định: 0

Long

Số nguyên có dấu. Một bit biểu diễn dấu (âm hoặc dương).

Kích thước: 4 byte
Trị: –2,147,483,648 đến 2,147,486,647
Trị mặc định: 0

Object

Chứa tham chiếu đến đối tượng (i.e., địa chỉ). Đối tượng có thể là OLE automation hay ActiveX component, hoặc đối tượng lớp bên trong ứng dụng.

Kích thước: 4 byte
Trị: tham chiếu đến đối tượng tùy ý.
Trị mặc định: Nothing

Single

Số thực dấu chấm động.
Kích thước: 4 byte
Trị:
- Trị âm: –3.402823E38 to –1.401298E-45
- Trị dương: 1.401298E-45 to 3.402823E38
Trị mặc định: 0

String (chiều dài cố định)

Được dùng trong VB khi bộ nhớ và dung lượng đĩa là mối quan tâm hàng đầu, nhưng nói chung ít được dùng.

Khai báo:

```
Dim|Private|Public varname As String * stringlength
```

Kích thước: Chiều dài của chuỗi
Trị: 1 đến 65,400 ký tự
Trị mặc định: Số khoảng trắng bằng với chiều dài chuỗi.

String (chiều dài thay đổi)

Kiểu dữ liệu chuỗi tự co dãn để có thể chứa đủ ký tự cần thiết, cho đến khoảng 2 tỉ.

Khai báo:

```
Dim variablename As String
```

VBA có nhiều hàm thao tác trên chuỗi. Danh sách các hàm này còn kéo dài hơn trong VB khi nhiều hàm được giới thiệu trong VBScript ngày nay cũng được chấp nhận trong VB.

Kích thước: 10 + chiều dài chuỗi
Trị: 0 đến 2 tỉ ký tự
Trị mặc định: chuỗi rỗng ("")

Kiểu người dùng

Kiểu người dùng cho phép chúng ta tạo kiểu dữ liệu đơn bằng cách kết hợp các kiểu dữ liệu bên trong VB, mảng, đối tượng, hay các kiểu người dùng khác.

Khai báo:

```
Type udtCustomer
   Name As String
   Code As Long
   Orders(20) As udtOrders
   RenewalDate As Date
End Type
```

Kiểu người dùng là cấu trúc dữ liệu quan trọng trong VB, chủ yếu được dùng khi giao tiếp với Windows API.

Kích thước: Tổng kích thước các thành phần bên trong.
Trị: Tùy theo trị các thành phần bên trong
Trị mặc định: Giá trị mặc định của các thành phần bên trong

Variant (ký tự)

Kiểu phụ rất giống kiểu chuỗi có chiều dài rhay đổi.
Các hàm về chuỗi của VB đều chấp nhập chuỗi variant, nhiều hàm có hai phiên bản trả về chuỗi hay chuỗi variant. Thí dụ hàm Left$ và Left.

Kích thước: 22 bytes + chiều dài chuỗi

Trị: Giống chuỗi có chiều dai thay đổi
Trị mặc định: Empty

Variant (số)

Bộ nhớ được cấp phát động tùy theo giá trị số. Variant cũng bao gồm kiểu phụ Decimal.

Kích thước: 16 bytes
Trị: Same as Double
Trị mặc định: Empty

Chuyển đổi kiểu

VBA cung cấp 2 tập hợp hàm để chuyển đổi. Tập hợp đầu, bao gồm *Int và Str*, có từ các phiên bản trước của VB và tồn tại chỉ để tượng thích ngược. Các hàm thuộc tập hợp thứ hai bắt đầu với chữ C là những hàm chuyển đổi mới hơn.
M$ khuyên chúng ta dùng tập hợp hàm thứ hai.

Cú pháp của tập hợp hàm thứ hai về cơ bản thì giống nhau. Thí dụ:

```
Cbool(tên_biến)
```

Các hàm chuyển đổi được VBA hỗ trợ là:

CBool

Chuyển 0 thành false, các trị khác thành true

CByte

Chuyển đổi *tên_biến* thành kiểu Byte. *tên_biến* có thể chứa dữ liệu số bất kỳ hay dữ liệu chuỗi có thể chuyển đổi thành số, trong khoảng 0 đến 255. Nếu *tên_biến* ngoài khoảng đó, VBA báo lỗi Overflow. Nếu *tên_biến* là dấu chấm động, nó được làm tròn thành số nguyên gần nhất trước khi chuyển đổi kiểu.

CDec

Chuyển đổi *tên_biến* thành kiểu phụ Decimal. Hàm này chấp nhận dữ liệu số hay chuỗi có thể chuyển thành số, trong giới hạn của kiểu Decimal. Đây là hàm cung cấp phương thức duy nhất để tạo kiểu phụ Decimal.

CDate

Chuyển đổi *tên_biến* thành kiểu Date. *CDate* chấp nhận số và chuỗi biểu diễn ngày và chuyển đổi thành dạng được mô tả bởi thông tin địa phương trên máy tính. Thí dụ máy tính được cài đặt dạng ngày của Mỹ mm/dd/yy, nếu ta nhập ngày dạng Anh dd/mm/yy và dùng hàm Cdate thì ngày được sẽ chuyển đổi sang dạng Mỹ.

CCur

Chuyển đổi *tên_biến* thành kiểu Currency. *CCur* chấp nhận dữ liệu số hay chuỗi bất kỳ biểu diễn giá trị tiền tệ. Hàm này nhận dạng thập phân hay dấu phân cách hàng ngàn theo thông tin địa phương của máy tính. Chỉ dùng cho VBA.

CDbl

Chuyển đổi *tên_biến* thành kiểu double. Chấp nhận dữ liệu số bất kỳ nằm trong giới hạn của kiểu Double hoặc bất kỳ dữ liệu chuỗi có thể chuyển đổi thành số trong giới hạn của kiểu Double.

CInt

Chuyển đổi *tên_biến* thành kiểu Integer.

CLng

Chuyển đổi *tên_biến* thành kiểu Long.

CSng

Chuyển đổi *tên_biến* thành kiểu Single.

CStr

Chuyển đổi *tên_biến* thành kiểu String. *CStr* chấp nhận kiểu dữ liệu bất kỳ.

CVar

Chuyển đổi *tên_biến* thành kiểu Variant. *CVar* chấp nhận kiểu dữ liệu bất kỳ.

Chuyển đổi kiểu ẩn trong VB

Chú ý rằng VB có nhiều cách chuyển đổi kiểu tự động. Thí dụ thuộc tính Text của text box rõ ràng có kiểu dữ liệu chuỗi và thuộc tính prompt của hộp thông báo cũng vậy. Như thế đoạn mã sau có lẽ sẽ gây lỗi:

```
Private Sub Command1_Click()
   1. Dim iValue As Integer
   2. iValue = txtTextBox.Text
   3. MsgBox Prompt:=iValue
End Sub
```

Tuy nhiên VB cho phép gán biểu diễn chuỗi của số cho kiểu dữ liệu Integer, rồi gán số này cho thuộc tính chuỗi prompt của hộp thoại. Chuyển đổi kiểu diễn ra tự động.

Variant

VBA chưa kiểu dữ liệu đặc biệt, Variant. Bên trong, kiểu Variant rất phức tạp nhưng cũng cực kỳ tiện dụng. Variant là kiểu dữ liệu mặc định của VBA. Thí dụ sau xem myVar là Variant:

```
Dim myVar
```

Kiểu dữ liệu Variant cho phép chúng ta dùng một biến với bất kỳ kiểu dữ liệu nội tại nào của VBA, kiểu dữ liệu gần nhất với giá trị được gán sẽ tự động làm việc.

Để minh họa, chúng ta khảo sát hai phiên bản của cùng một hàm:

```
1.  Private Function GoodStuff(vAnything, vSomething,
    vSomethingElse)
2.     If vAnything > 1 And vSomething > "" Then
3.        GoodStuff = vAnything * vSomethingElse
```

```
4.     Else
5.        GoodStuff = vAnything + 10
6.     End If
7.  End Function

8.  Private Function GoodStuff(iAnything As Integer, sSomething As_
    String, iSomethingElse As Integer) As Integer
9.     If iAnything > 1 And sSomething > "" Then
10.       GoodStuff = iAnything * iSomethingElse
11.    Else
12.       GoodStuff = iAnything + 10
13.    End If
14. End Function
```

Các kiểu phụ variant đặc biệt

Thêm vào các kiểu dữ liệu nội tại như kể trên, variant cũng hỗ trợ các kiểu dữ liệu đặc biệt:

Empty

Kiểu phụ Empty được tự động gán cho biến Variant vừa khai báo, trước khi ta gán trị cho nó. Thí dụ:

```
Dim var1, var2
var2 = 0
```

Kiểu phụ của var1 là Empty, trong khi var2 chỉ Empty trong khoảng thời gian ngắn giữa việc thực thi mệnh đề Dim của dòng thứ nhất và phát biểu gán ở dòng thứ hai.

Thêm nữa, kiểu phụ của biến là Empty nếu biến được gán một cách tường minh bằng trị Empty.

```
Dim var1
var1 = Empty
```

Null

Null là kiểu phụ đặc biệt chỉ rằng biến không chứa trị hợp lệ. Thông thường biến được gán trị Null khi có một lỗi xảy ra.
Biến phải được gán trị Null một cách tường minh.

```
var1 = Null
```

Trị Null cũng là kết quả của phép toán trong đó một trong những giá trị của biểu thức là Null.

```
dim myVarOne, myVarTwo, myVarThree 'Cả 3 biến là EMPTY
myVarOne = 9
myVarTwo=NULL 'Cho biến này là NULL
myVarThree = myVarOne + myVarTwo 'Kết quả là NULL
```

Error

Kiểu phụ Error được dùng lưu trữ mã lỗi. Các mã lỗi được VBA sinh tự động, được dùng trong các đoạn chương trình kiểm soát lỗi.

Xác định kiểu phụ variant

Cũng rất tốt khi kiểu dữ liệu variant quản lý tất cả dữ liệu chúng ta nhập, nhưng chính xác một biến đang có kiểu dữ liệu nào? VBA cung cấp hai hàm: *VarType*, trả lại một số nói lên kiểu dữ liệu; và *TypeName*, trả lại chuỗi tên kiểu.

VarType

Cú pháp của *VarType* là:

```
VarType(tên_biến)
```

Bảng sau đây liệt kê các trị được *VarType* trả về và các hằng VBA tượng ứng:

Value	Data Subtype	VBA Constant
0	Empty	vbEmpty
1	Null	vbNull
2	Integer	vbInteger
3	Long Integer	vbLong
4	Single	vbSingle
5	Double	vbDouble
6	Currency	vbCurrency
7	Date	vbDate
8	String	vbString
9	OLE Automation Object	vbObject
10	Error	vbError
11	Boolean	vbBoolean
12	Array of Variant	vbVariant
13	Data access object	vbDataObject
14	Decimal	vbDecimal
17	Byte	vbByte
36	User-defined Type	vbUserDefinedType
8192	Array	vbArray

Tuy nhiên, hàm *VarType* không bao giờ trả về 8192 như trong bảng, điều này chỉ nói lên sự tồn tại của mảng. Khi truyền cho một mảng, *VarType* trả về 8192 cộng với giá trị của kiểu phần tử mảng. Thí dụ, khi truyền mảng các chuỗi cho *VarType*, trị trả về là 8200 (8192 + 8).

TypeName

Hàm *TypeName* trả lại tên kiểu phụ thay vì số. Cú pháp của *TypeName* là:

```
result = TypeName(variable)
```

Giống như hàm *VarType*, *TypeName* là chỉ đọc, dùng để xác định kiểu phụ của biến nhưng không thể dùng để đặt kiểu cho biến. Để làm điều này, chúng ta phải dùng hàm chuyển đổi kiểu.

Return Value	Data Subtype
<object type>	Actual type name of an object
Boolean	Boolean value: True or False
Byte	Byte value
Currency	Currency value
Date	Date or time value
Decimal	Decimal (single-precision) value
Double	Double-precision floating-point value
Empty	Uninitialized
Error	Error
Integer	Integer value

Long	Long integer value
Nothing	Object variable that doesn't yet refer to an object instance
Null	No valid data
Object	Generic object
Single	Single-precision floating-point value
String	Character string value
Variant()	Variant array
Unknown	Unknown object type

Nếu chúng ta truyền một mảng của một kiểu dữ liệu cụ thể cho hàm *TypeName*, chuỗi trả về là chuỗi của kiểu dữ liệu đó cùng với "()" để ám chỉ mảng. Thí dụ với bioến là mảng các chuỗi, trị trả về là "String()".

Để đoạn mã trở nên dễ đọc và dễ bảo trì, có thể dùng các hàm trên như sau:

```
If TypeName(x) = "Double" Then
```

Dữ liệu Variant và dữ liệu kiểu khác

Variant có vẻ là câu trả lời cho tất cả yêu cầu về kiểu dữ liệu, nhưng cũng có giá phải trả. Variant còn hơn là một kiểu dữ liệu, tự nó là một ứng dụng. Nó phải qua nhiều xử lý để xác định được kiểu dữ liệu của một giá trị trừu tượng. Một biểu thức chỉ có trị variant xử lý chậm hơn 33% so với cùng biểu thức dùng kiểu dữ liệu nội tại.

Hàm về variant và hàm về các kiểu khác

Ngôn ngữ VBA bao gồm một số hàm xử lý chuỗi với hai phiên bản, một trả về variant và một trả về chuỗi. Loại hàm thứ hai lấy tên loại hàm thứ nhất cộng thêm ký tự $ chỉ chuỗi (thí dụ, Left và Left$).

Khảo sát phiên bản variant

```
1. Dim sString
2. Dim sPartString
3. sString = "ABCDEFGH"
4. sPartString = Mid(sString, 1, 2)
```

và phiên bản chuỗi

```
1. Dim sString As String
2. Dim sPartString As String
3. sString = "ABCDEFGH"
4. sPartString = Mid$(sString, 1, 2)
```

Ta nhận thấy phiên bản variant chậm hơn 50%. Hiển nhiên đây là sự khác biệt có ý nghĩa, gợi ý chúng ta dùng phiên bản có kiểu bất cứ khi nào có thể.

Khai báo biến và hằng

Như đã biết, VBA hỗ trợ kiểu dữ liệu mặc định, vì vậy không giống như nhiều ngôn ngữ khác, VBA cho phép khai báo biến ẩn. Ngay khi chúng ta dùng tên biến hay tên hằng trong mã lệnh, VBA làm tất cả việc cần thiết để cấp phát bộ nhớ, … và biến xem như đã được khai báo.

Tuy nhiên, kinh nghiệm lập trình cho thấy nên khai báo biến và hằng tường minh bằng cách dùng phát biểu Dim, Private, hay Public statements.

Cú pháp:

```
Dim VariableName As datatype
Private VariableName As datatype
Public VariableName As datatype
```

Nếu có nhiều biến cần khai báo, có thể viết cùng dòng phân cách bởi dấu phẩy:

```
Dim iRefNo As Integer, iAnyVar As Integer
```

Bằng cách khai báo rõ biến theo cách này, chúng ta có thể giảm số lỗi trong chương trình do gõ sai tên biến.

Option Explicit

Dùng phát biểu Option Explicit là một kinh nghiệm hay. Nó buộc chúng ta khai báo rõ biến và hằng. Chúng ta có thể để VB tự động thêm nó vào các modun mới tạo bằng cách đánh dấu option Require Variable Declaration trong tab Editor của hộp thoại Options.
Khi Option Explicit được dùng, VB sinh lỗi compile-time khi nó gặp một biến chưa được khai báo..

Dữ liệu rỗng

Một thành phần quan trọng của ngôn ngữ lập trình là khả năng phát hiện và quảlý dữ liệu rỗng. Chúng ta muốn nói đến dữ liệu chưa có. VBA phát triển một số cách cho phép chúng ta gán trị empty hay null cho biến. Hiểu được sự khác biệt giữa chúng là quan trọng vì mỗi thứ đều có cách dùng riêng và không thể thay thế cho nhau.

vbNull

Dùng với hàm *VarType* để xác định biến chứa Null. Thí dụ:

```
varValue = Null
If VarType(varValue) = vbNull Then
```

Chú ý là không thể dùng hằng vbNull để gán trị Null.

vbNullChar

Dùng để gán hay test ký tự null chr(0). Nói cách khác vbNullChar tượng đương chr(0).

```
sMyString & vbNullChar
```

vbNullString

Gán hoặc test chuỗi chiều dài 0 (chuỗi rỗng).

```
strVar1 = vbNullString
```

tượng đương với:

```
strVar1 = ""
```

Từ khóa Null

Gán trị Null cho biến variant. Có thể test trị null của biến bằng hàm *IsNull*. Chú ý rằng đoạn mã sau

```
varValue = Null
if varValue = Null
```

trả về False, vì Null là False và vì vậy bất kỳ biểu thức nào chứa Null cũng trả về False.

Đoạn mã sau chỉ ra khi nào dùng và khi nào không dùng từ klhóa Null

```
1. Dim i As Variant
2. i = Null
3. If i = Null Then
4.    MsgBox "It's null" 'Sai
5. End If
6. If IsNull(i) Then
7.    MsgBox "It's null" 'Đúng
8. End If
```

Chú ý là từ khóa Null không thể dùng để gán cho biến kiểu bình thường.

vbEmpty

Xác định biến variant được khởi tạo hay chưa. Thí dụ:

```
If IsEmpty(varValue) Then
```

Giống với:

```
If varValue = vbEmpty then
```

Tuy nhiên không thể dùng vbEmpty để gán trị empty cho biến variant.

Nothing keyword

Chỉ dùng với biến đối tượng để xác định biến đã tham chiếu đến một đối tượng.

```
If objVar Is Not Nothing Then
```

Hoặc để hủy tham chiếu đến đối tượng:

```
Set objvar = Nothing
```

Array Variables

Tạo một mảng được gọi là *dimensioning* một mảng (nghĩa à xác định kích thước của nó). Các phần tử dữ liệu bên trong được gọi là *phần tử* và số dùng truy cập phần tử được gọi là *chỉ số*. Các chỉ số nhỏ nhất và lớn nhất được gọi là *cận*.

Trong VBA, có bốn kiểu mảng: mảng cố định hay mảng động, mảng một chiều hay nhiều chiều.

Mảng cố định

Hầu như chúng ta luôn biết có bao nhiêu phần tử cần được lưu trong mảng, vì vậy có thể định ra kích thước thích hợp khi khai báo mảng.

```
Dim myArray(5) As Integer
```

Dòng trên khai báo mảng tên là myArray với 6 phần tử bắt đầu từ vị trí 0.

Chúng ta có thể tạo mảng cùng với các phần tử của nó bằng hàm Array:

```
myArray = Array(12,3,13,64,245,75)
```

Mảng động

Trong trường hợp không biết trước số phần tử cần lưu trong mảng, chúng ta dùng mảng động. Mảng động cho phép chúng ta mở rộng số phần tử mảng khi chương trình đang hoạt động bằng phát biểu ReDim.

Mảng động được khai báo bằng cách bỏ qua số phần tử mảng:

```
Dim iDynamicArray() As Integer
```

Khi cần định lại kích thước mảng, dùng từ khóa ReDim:

```
ReDim iDynamicArray(10)
```

Chúng ta cũng có thể khai báo mảng động cùng với số phần tử khởi tạo bằng cách dùng ReDim:

```
ReDim anyDynamicArray(4) As Integer
```

Không có hạn chế về số lần định lại kích thước mảng động, nhưng mỗi lần như vậy dữ liệu lưu trong mảng bị mất. Nếu cần giữ lại dữ liệu đã có, dùng từ khóa Preserve:

```
ReDim Preserve myDynamicArray(10)
```

Trên thực tế, ReDim tạo mảng mới, Preserve sao chép dữ liệu từ mảng cũ sang mảng mới. Điều quan trọng cần nhớ là khi giảm kích thước mảng, ta sẽ mất dữ liệu thuộc các phần tử bị xóa.

Trong khi có thể định lại kích thước mảng bằng cách điều chỉnh cận trên, chúng ta không thể điều chỉnh cận dưới.

Redim một mảng luôn làm chậm tốc độ thực thi, vì vậy chúng ta dùng mảng cố định mỗi khi có thể. Một khi đã dùng mảng động, chúng ta nên điều chỉnh kích thước mảng mỗi lần nhiều phần tử thay vì một.

```
If lngCurPtr > UBound(varArray) Then
ReDim Preserve varArray(UBound(varArray) + 10)
End If
```

```
1.  Option Explicit 'require variable declaration
2.  ReDim sMyArray(0) As String 'create a 1-element dynamic array
3.  Dim iIndex As Integer 'variable to track array index
4.  iIndex = 0 'assign the first index number
5.  Sub cmdButton1_OnClick
6.  'Store the user input in the array
7.     sMyArray(intIndex) = txtText1.Text
8.  'increment the array counter by one
9.     iIndex = iIndex + 1
10. 'increase the size of the array
11.    ReDim Preserve sMyArray(iIndex)
12.    txtText1.Text = "" 'Empty the text box again
13. End Sub
```

Xác định cận của mảng

Các hàm *Ubound* và *LBound* có thể tìm cận trên và dưới của một mảng.
Cú pháp:

```
x = UBound(arrayname)
```

UBound trả lại chỉ số cao nhất của mảng. Số phần tử thật sự của mảng phụ thuộc vào điểm bắt đầu của mảng. Nếu dùng cận dưới mặc định của mảng là 0, Ubound nhỏ hơn số phần tử 1.

```
iArraySize = UBound(array) + 1
```

Tuy nhiên, công thức chính xác là:

```
iArraySize = UBound(array) – LBound(array) + 1
```

Hàm *UBound* thường dùng với mảng động:

```
1. Option Explicit
2. Private sValues() As String

3. Private Sub Form_Load()
```

```
4.     ReDim sValues(0)
5. End Sub

6. Private Sub Command1_Click()
7.     sValues(UBound(sValues)) = txtTextBox.Text
8.     ReDim Preserve sValues(UBound(sValues) + 1)
9. End Sub
```

Chú ý là dùng hàm *UBound* trên một mảng chưa khởi tạo sẽ gây lỗi Out of Range.

Đặt cận dưới

Chúng ta có thể thay đổi cận dưới mặc định trên từng modun bằng phát biểu Option Base ở phần khai báo của modun. Thí dụ:

```
Option Base 1
```

Sinh ra các mảng có phần tử đầu 1. Phát biểu Option Base phải dùng trong modun trước bất kỳ khai báo biến nào.

Một cách khác là đặt cả cận trên và dưới trong khai báo biến:

```
Dim arrayname(lowerboundary To upperboundary) As datatype
```

Mảng nhiều chiều

Trong mảng một chiều, dữ liệu lưu trữ bên trong không có cấu trúc; nó được truy cập liên tiếp, có một mẫu dữ liệu cho mỗi phần tử. Khi chúng ta cần lưu trữ nhiều hơn một mẫu dữ liệu cho một phần tử, chúng ta dùng hoặc là mảng nhiều chiều hoặc là dữ liệu kiểu người dùng định nghĩa.

Mảng nhiều chiều cho phép chúng ta có một mảng riêng cho mỗi phần tử của mảng. Cấu trúc của mảng nhiều chiều giống như bảng CSDL. Dòng của bảng tượng trưng cho chiều thứ nhất, cột của bảng tượng trưng cho chiều thứ hai.

Mảng nhiều chiều có tối đa 60 chiều, mặc dù chúng ta hiếm khi dùng nhiều hơn hai hay ba chiều.

Để định nghĩa mảng nhiều chiều, dùng cú pháp sau:

```
Dim arrayname(upperboundDimension1, _
upperboundDimension2, ....) As Datatype
```

Cũng như với mảng một chiều, chúng ta có thể chỉ định cận dưới trong định nghĩa mảng:

```
Private myArray(1 To 20, 0 To 50) As String
```

Mảng động nhiều chiều

Giống như mảng một chiều, mảng nhiều chiều có thể là động và qui luật định lại kích thước cũng tượng tự.

Các qui tắc dùng cho mảng động nhiều chiều là:

- Có thể ReDim để thay đổi cả về số chiều lẫn kích thước mỗi chiều.

```
1. Private myArray() As Integer
2. Private Sub cmdButtonOne_OnClick
3.   ReDim myArray(10,5)
4. End Sub
5. Private Sub cmdButtonTwo_OnClick
6.   ReDim myArray(4,10,2)
7. End Sub
```

- Nếu dùng từ khóa Preserve, chỉ có thể điều chỉnh kích thước của chiều sau cùng của mảng và không thể thay đổi số chiều:

```
...
ReDim myArray(10,5,2)
...
ReDim Preserve myArray(10,5,4)
...
```

Trong trường hợp mảng nhiều, hàm *UBound* cần thêm tham số chỉ định chiều:

largestElement = UBound(*arrayname*, *dimensionNo*)

Tượng tự với hàm *LBound*:

smallestElement = LBound(*arrayname*, *dimensionNo*)

Kiểu người dùng định nghĩa

Một hạn chế chính của mảng nhiều chiều là tất cả các chiều của mảng phải có cùng kiểu dữ liệu. Kiểu người dùng định nghĩa (UDT), kết hợp nhiều kiểu dữ liệu vào một kiểu dữ liệu mới, vượt qua hạn chế này.

Đoạn mã sau định nghĩa một UDT đơn giản:

```
1. Private Type custRecord
2.   custAccNo As Long
3.   custName As String
4.   RenewalDate As Date
5.   End Type
6. Private custArray(10) As custRecord
```

Dòng cuối cùng tạo một mảng cục bộ của UDT.

Ta cũng có thể dùng một UDT khác trong một UDT:

```
1.  Private Type custOrders
2.    OrderNo As Long
3.    OrderDate As Long
4.  End Type

5.  Private Type custRecord
6.    custAccNo As Long
7.    custName As String
8.    RenewalDate As Date
9.    orders(10) As custOrders
10. End Type
11. Private custArray(10) As custRecord
```

Sau đây là thí dụ về truy cập dữ liệu trong UDT:

```
Text1.Text = custArray(iCust).custName
Text2.Text = custArray(iCust).orders(iOrder).OrderNo
```

Phạm vi (tầm) và đời sống của biến

Đôi khi chúng ta cần một biến được thấy trong tất cả thủ tục trong khi biến khác chỉ có giá trị trong một thủ tục nào đó. Khả năng được nhìn thấy của biến gọi là phạm vi (tầm). Gắn liền với tầm là đời sống của biến, hay là chu kỳ của việc thực thi chương trình khi biến tồn tại và có giá trị sử dụng. Chính xác nơi chúng ta khai báo một biến hay hằng trong chương trình xác định tầm và thời gian sống của biến.

Biến khai báo trong phần khai báo của modun bằng các sử dụng từ khóa Private có thể được truy cập bởi tất cả thủ tục trong modun.

Biến khai báo trong phần khai báo của modun bằng các sử dụng từ khóa Public có thể được truy cập trong toàn project.

Biến khai báo trong phần khai báo của modun lớp bằng các sử dụng từ khóa Public có thể được truy cập trong toàn project một khi đối tượng của lớp được tạo.

Biến khai báo bằng cách dùng phát biểu Dim trong thủ tục hay hàm chỉ có thể truy cập từ bên trong thủ tục hay hàm đó.

Biến trong phạm vi thủ tục

Biến khai báo trong thủ tục chỉ có thể dùng trong thủ tục. Ta có thể dùng cùng tên biến trong các thủ tục khác nhau. Biến tầm mức thủ tục kết thúc khi phát biểu End Sub hay End Function được thi hành.

Có một loại biến mức thủ tục đặc biệt là biến s*tatic*, mặc dù khai báo trong thủ tục và chỉ được tham chiếu trong thủ tục nhưng có thời gian sống ở mức modun. Giá trị của biến vẫn được lưu giữ qua các lần gọi thủ tục.

Khai báo bằng từ khóa Static:

```
Static lngExecuted As Long
```

Cũng có thể khai báo một thủ tục là Static, khi đó tất cả biến trong thủ tục đều là Static.

```
Static Procedure MyProcedure()
Dim iCtr As Integer
```

Trong thủ tục, chỉ có thể dùng Dim hay Static để khai báo biến, không thể khai báo biến hay hằng bằng Public, Private, hay Friend.

Biến trong phạm vi modun

Biến có phạm vi modun có thể được truy cập bởi bất cứ hàm thay thủ tục nào trong modun. Biến và hằng mức modun tồn tại trong bộ nhớ suốt thời gian tồn tại của modun. Để tạo biến hay hằng mức modun, dùng phát biểu Dim hay Private trong phần khai báo của modun.

Friend Scope

Từ khóa Friend chỉ có thể dùng cho khai báo biến hay thủ tục bên trong modun đối tượng như một lớp hay một modun form. Friend cho phép các modun đối tượng khác trong cùng một project truy cập biến và phương thức mà không cần phải khai báo như Public.

Public Scope

Dùng bên ngoài thủ tục thay cho phát biểu Dim, Public cho phép biến được nhìn thấy bởi tất cả thủ tục trong tất cả modun của project. Nếu được dùng trong Class module, phạm vi của nó vượt ra ngoài biên giới của project hiện hành. Việc tạo tự động giao tiếp COM cho một thủ tục hay thuộc tính public bất kỳ có nghĩa là nó có thể được gọi từ các thành phần software khác như là một phương thức hay thuộc tính của lớp trong đó nó được định nghĩa.

Biến đối tượng và sự liên kết

Khai báo biến đối tượng

Biến đối tượng được khai báo khá giống với các biến kiểu khác. Sau đây là ba cách khai báo:

```
1. Dim myObject As LibName.ClassName
2. Dim myObject As New LibName.ClassName
3. Dim myObject As Object
```

Trong các cách trên, phát biểu Private hay Public có thể thay cho phát biểu Dim, phạm vi của biến cũng giống như các kiểu biến khác.

Trong khai báo đầu tiên, biến đối tượng được tham chiếu đến class type library, nhưng không có instance nào của lớp được gán cho biến. Lúc này, myObject được đặt Nothing. Để tham chiếu lớp theo cách này, ta phải dùng hộp thoại References. Để gán tham chiếu đến một thể hiện của lớp, phải dùng phát biểu Set trước tên biến:

```
Set myObject = LibName.ClassName
```

Cách này tạo ra một tham chiếu liên kết sớm tới đối tượng.

Trong phát biểu thứ hai, tham chiếu đến một thể hiện mới của lớp được gán cho biến đối tượng. Một lần nữa, ta phải dùng đến hộp References. Phương pháp thứ hai cũng tạo mộ tham chiếu liên kết sớm đến đối tượng, tuy nhiên đối tượng chưa thật sự được tao ra cho đến khi biến đối tượng được dùng.

Trong phát biểu thứ ba, biến đối tượng được khai báo thuộc kiểu đối tượng tổng quát. Cách này thường dùng khi chúng ta không biết trước kiểu đối tượng chúng ta sẽ tạo. Lúc này, biến đối tượng cũng có giá trị Nothing. Để gán một tham chiếu đối tượng cho nó, chúng ta phải dùng hoặc hàm *CreateObject* hoặc *GetObject*. Biến đối tượng khai báo theo cách này được gọi là liên kết muộn.

Liên kết sớm và liên kết muộn

Liên kết muộn kém hiệu quả hơn liên kết sớm, nhưng không phải luôn luôn như vậy. Có một số yếu tố cần xem xét khi chọn phương thức liên kết của đối tượng.
Thứ nhất, đối tượng mà ta muốn liên kết thực thi trong cùng tiến trình của client hay chạy trong tiến trình riêng của nó? Nó sẽ chạy trên cùng một máy hay từ server ở xa? Nói chung liên kết muộn hiệu quả hơn một tí với những out-of-process ActiveX EXEs, và liên kết sớm hiệu quả hơn nhiều với những in-process DLLs.
Yếu tố thứ hai ảnh hưởng lên hiệu suất của liên kết sớm hay muộn là hệ điều hành. Sự khác biệt về hiệu suất thấy rõ trên Windows 95, trong khi ít khác biệt trên Windows NT.

Đối tượng Collection

Đối tượng collection là phiên bản hướng đối tượng của mảng VB, có thể chứa bất kỳ kiểu dữ liệu nào. Collection hỗ trợ bốn phương thức sau:

Add

Thêm một phần tử vào collection. Cùng với dữ liệu, chúng ta có thể cho một khóa để có thể truy cập phần tử từ collection.

Count

Trả lại số phần tử trong collection.

Item

Lấy ra một phần tử từ collection hoặc bằng chỉ số hoặc bằng khóa của nó.

Remove

Xóa phần tử khỏi collection hoặc bằng chỉ số hoặc bằng khóa của nó.

```
Dim colStates As New Collection
colStates.Add "New York", "NY"
colStates.Add "Michigan", "MI"
```

Giống như mảng, các phần tử của collection có thể được duyệt qua bằng vòng lặp For Each...Next, và giống như mảng, chúng có thể được truy cập bằng chỉ số, mặc dầu cận dưới của đối tượng collection luôn là 1 và không thể đặt lại qua code.

Khác biệt so với mảng là collection dễ truy cập và bảo dưỡng:

• Phần tử mới có thể chèn vào trước hay sau phần tử đã có trên cơ sở chỉ số hay khóa.
• Phần tử có thể được lấy ra bằng chỉ số hay khóa.
• Phần tử có thể được xóa dựa trên chỉ số hay khóa. Tuy nhiên xóa nhiều phần tử dựa trên chỉ số phải theo thứ tự ngược vì collection đánh chỉ số lại sau mỗi lần xóa.

Truyền tham biến

VB cho phép truyền biến giữa các thủ tục và component theo hai cách. Bên trong định nghĩa hàm hay thủ tục, chúng ta dùng hoặc ByRef hoặc ByVal cho mỗi biến trong danh sách tham đối.

ByRef

Đây là phương thức mặc định để truyền biến giữa các thủ tục trong VB, nghĩa là nếu không chỉ định rõ ByVal hay ByRef, VB xem như ByRef.

ByRef nghĩa là biến thuyền qua tham chiếu, nói cách khác chỉ có tham chiếu đến biến gốc được truyền cho thủ tục được gọi chứ không phải là bản sao của biến. Hậu quả là biến có thể bị thay đổi từ bên trong thủ tục được gọi.

```
1.  Private Sub Command1_Click()
2.    Dim blnCancel As Boolean
3.    Dim lReturn As Long
4.    lReturn = GetValue(blnCancel)
5.    If blnCancel Then
6.      Exit Sub
7.    Else
8.      MsgBox lReturn
9.    End If
10. End Sub
11. Private Function GetValue(ByRef Cancel As Boolean) As Long
12.   Dim sResponse As String
13.   Dim iResponse As Integer
14.   Cancel = False
15.   sResponse = InputBox(Prompt:="Enter a value", _
16.   Title:="Input Required", Default:=0)
17.   'an inputbox returns a zero length string if _
18.   the Cancel button was clicked
19.   If sResponse = "" Then
20.     Cancel = True
21.   Else
22.     If IsNumeric(sResponse) Then
23.       GetValue = CLng(sResponse)
24.     End If
25.   End If
26. End Function
```

ByVal

Dùng từ khóa ByVal để truyền biến bằng trị, thủ tục được gọi nhận bản sao của biến. Sự thay đổi của bi bên trong thủ tục không ảnh hưởng đến biến gốc.

ByRef và ByVal

Khi truyền biến giữa các thủ tục trong cùng project hay giữa các phương thức của ActiveX component, ByRef nhanh hơn ByVal nhiều. Tuy nhiên khi truyền biến cho các phương thức trong out-of-process server, ByVal nhanh hơn vì các thủ tục trong các process khác nhau không thể chia sẽ bộ nhớ.

Tham biến tùy chọn

Từ khóa Optional có thể dùng trong danh sách tham biến khi khai báo một thủ tục để chỉ rằng tham biến đó có thể không cần được truyền. Một hạn chế là tất cả tham đối đi sau tham đối option đầu tiên cũng phải là option.

Để kiểm tra xem tham biến option kiểu variant có được truyền cho thủ tục hay không, dùng hàm IsMissing. Các kiểu dữ liệu khác sẽ có trị mặc định khi nó không được truyền như tham biến. Tuy nhiên điều này có thể gây nhầm lẫn. Tham biến option kiểu số nguyên không được truyền thì trong thủ tục nó sẽ có trị 0, giống như khi nó được truyền băng trị 0!

ParamArray

Từ khóa ParamArray (viết tắt của Parameter Array) cho phép chúng ta chấp nhận một số tùy ý tham biến. ParamArray phải là tham biến cuối cùng ttrong danh sách và không được dùng trong cùng danh sách tham biến option.

ParamArray là mảng variant tùy chọn. Nghĩa là mảng có thể rỗng hoặc chứa một số tùy ý các phần tử kiểu variant.

```
1.  Private Sub cmdCallDoStuff_Click()
2.    Dim blnOK As Boolean
3.    blnOK = DoStuff("Wednesday", 1234, _
4.    CDate("04/12/1999"), 123.444)
5.  End Sub
6.  Private Sub cmdCallDoOtherStuff_Click()
7.    Dim blnOK As Boolean
8.    Dim oTest As testEXE.txtClass
9.    Set oTest = New testEXE.txtClass
10.   blnOK = DoStuff(123, 9999999.99, "Hello World", oTest)
11.   Set oTest = Nothing
12. End Sub
13. Private Function DoStuff(ParamArray anyArgs()) As Boolean
14.   Dim i As Integer
15.   For i = 0 To UBound(anyArgs)
16.     MsgBox anyArgs(i) & vbCrLf & TypeName(anyArgs(i))
17.   Next i
18. End Function
```

Hằng nội tại

Cùng với việc cho phép định nghĩa hằng bằng từ khóa Const, VBA có một số hằng đã được định nghĩa sẵn. Thí dụ, thay vì viết

```
If myObject.ForeColor = &hFFFF Then
```

Ta có thể viết:

```
If myObject.ForeColor = vbYellow Then
```

Có thể tìm thông tin về hằng định nghĩa sẵn trong VB object browser.

CHƯƠNG II. Các Phát biểu và hàm chuẩn trong VBA

Cấu trúc	*Mục đích*
Phát biểu **GoTo**	Nhảy đến một phát biểu cụ thể
Cấu trúc **If – Then**	Thực hiện một điều gì đó nếu điều gì đó đúng
Select Case	Thực hiện một vài điều, tuỳ thuộc vào một vài giá trị
Vòng lập **For – Next**	Thực hiện một số phát biểu với số lần cụ thể.
Vòng lập **Do – While**	Thực hiện điều gì đó cho trong khi điều kiện nào đó vẫn đúng.
Vòng lập **Do – Until**	Thực hiện điều gì đó cho tới khi điều kiện nào đó đúng

I. CÁC PHÁT BIỂU ĐIỀU KHIỂN

a. **Cấu trúc chọn lựa IF**:

IF < Biểu thức Logic điều kiện> **THEN**
‘Khi điều kiện đúng
‘Nội dụng các câu lệnh cần thực hiện
ELSE
‘Khi điều kiện sai
‘Nội dụng các câu lệnh cần thực hiện
END IF

Trong câu lệnh không nhất thiết phải sử dụng ELSE, có thể bỏ qua nó tùy vào mục đích
Một số ví dụ:

```
Sub Hello()
   If ThoiGian<0.5 Then MsgBox "Chào buổi sáng"
End Sub


Sub Hello()
   If ThoiGian < 0.5 Then MsgBox "Chào buổi sáng"
   If ThoiGian >= 0.5 Then MsgBox "Chào buổi chiều"
End Sub
```

Các bạn cũng có thể kết hợp ví dụ trên như sau:

```
Sub Hello()
   If ThoiGian<12 Then MsgBox "Chào buổi sáng" Else _
      MsgBox “Chào buổi chiều”
End Sub
```

Bạn chú ý là trong ví dụ trên phát biểu If-Then-Else trên cùng một dòng. Các bạn cũng có thể viết lại như sau:

```
Sub Hello()
   If ThoiGian<12 Then
      MsgBox "Chào buổi sáng"
      ‘Các bạn có thể có nhiều câu lệnh ở đây
   Else
      MsgBox “Chào buổi chiều”
      ‘Các bạn có thể có nhiều câu lệnh ở đây
   End If
End Sub
```

b. Cấu trúc SELECT CASE :

SELECT CASE <Biến hay một biểu thức>
CASE <giá trị nhất của biến hay của 1 biểu thức>

CASE <giá trị hai của biến hay của 1 biểu thức>
…
CASE <giá trị n của biến hay của 1 biểu thức>
CASE ELSE
‘Khi tất cả các giá trị ở trên đều không đúng
‘Nội dung các lệnh cần thực hiện
END SELECT
Tương tự như IF trong câu lệnh không nhất thiết phải sử dụng CASE ELSE, có thể bỏ qua nó tùy vào mục đích.

Ví dụ dùng cấu trúc **Select-Case**

```
Sub ShowDiscount3()
   Dim Quantity As Integer
   Dim Discount As Double
   Quantity = InputBox("Enter Quantity: ")
   Select Case Quantity
   Case 0 to 24
        Discount=0.1
   Case 25 To 49
        Discount=0.15
   Case 50 To 74
        Discount=0.2
   Case Is >=75
        Discount=0.25
   End Select
   MsgBox "Discount: " & Discount
End Sub
```

Trong ví dụ trên chúng ta sẽ xét biến Quantity. Trong trường hợp có giá trị từ 0 đến 24 thì ta cho Discount=0.1, từ 25 đến 49 ta cho giá trị Discount =0.15, từ 50 đến 74 ta cho giá trị Discount=0.2, nếu trên hay bằng 75 ta cho giá trị Discount=0.25.

II. CÁC CẤU TRÚC LẶP :

a. Cấu trúc DO WHILE ... LOOP :

DO WHILE <Biểu thức điều kiện>
‘Các câu lệnh muốn thực thi biểu thức điều kiện còn đúng
LOOP ‘Quay về DO WHILE để kiểm tra biểu thức điều kiện

Khi VB thực hiện vòng lặp này, đầu tiên sẽ kiểm tra biểu thức điều kiện. Nếu Sai, nó sẽ dừng lại vòng lặp ngay và thực hiện câu lệnh kế tiếp còn nếu đúng thì thực hiện các lệnh bên trong. Vậy cấu trúc DO WHILE… LOOP thực hiện các câu lệnh bên trong nó khi điều kiện đúng.

b. Cấu trúc DO … LOOP WHILE :

DO
‘Các câu lệnh thực thi
LOOP WHILE <Biểu thức điều kiện> ‘Quay về DO nếu điều kiện đúng
Khi VB thực hiện vòng lặp này, đầu tiên sẽ thực hiện khối lệnh bên trong nó ngai. Sau khi thực hiện nó sẽ kiểm tra điều kiện. Nếu đúng sẽ quay lại còn sai thì dừng vòng lặp.

Xét hai ví dụ sau:

```
Sub DoWhileDemo()
```

```
   Do While ActiveCell.Value <> Empty
      ActiveCell.Value = ActiveCell.Value * 2
      ActiveCell.Offset(1, 0).Select
   Loop
End Sub
Sub DoLoopWhileDemo()
   Do
      ActiveCell.Value = ActiveCell.Value * 2
      ActiveCell.Offset(1, 0).Select
   Loop While ActiveCell.Value <> Empty
End Sub
```

Do-While loop bao giờ cũng thực hiện kiểm tra điều kiện trước khi thực hiện các phát biểu bên trong vòng lặp.
Do-Loop While ngược lại thực hiện kiểm tra điều kiện sau khi đã thực hiện các phát biểu bên trong vòng lặp.

c. Cấu trúc DO … LOOP UNTIL :

DO

‘Các câu lệnh thực thi

LOOP UNTIL <Biểu thức điều kiện> ‘Quay về DO nếu điều kiện sai

Giống như DO … LOOP WHILE nhưng nó sẽ thoát khỏi vòng lặp khi điều kiện Đúng

Vòng lặp Do-Until có cấu trúc gần giống với cấu trúc Do-While. Nhưng Do-Until loop, sẽ thực hiện cho tới khi điều kiện đúng. Các bạn hãy xem hai ví dụ sau đây, các bạn có nhận xét gì?

```
Sub DoUntilDemo()
   Do Until IsEmpty(ActiveCell.Value)
      ActiveCell.Value = ActiveCell.Value * 2
      ActiveCell.Offset(1, 0).Select
   Loop
End Sub
Sub DoLoopUntilDemo()
   Do
      ActiveCell.Value = ActiveCell.Value * 2
      ActiveCell.Offset(1, 0).Select
   Loop Until IsEmpty(ActiveCell.Value)
End Sub
```

d. Cấu trúc **FOR … NEXT**

FOR <Biến = Giá trị đầu> **TO** <Giá trị cuối> [STEP khoảng tăng]

‘ Phần các lệnh thực thi khi biến chưa đạt giá trị cuối

NEXT Biến

(Phần Step có thể có hoặc không, VB ngầm hiểu là +1)
Cấu trúc này lặp với số lần biết trước, lặp từ Giá trị đầu đến giá trị cuối (giá trị đầu có thể lớn hơn giá trị cuối nếu step <0)

Ví dụ về vòng lặp For-next:

```
Sub FillRange()
   Dim Count As Integer
   For Count = 1 To 100
      ActiveCell.Offset(Count - 1, 0) = Rnd
      'Rnd hàm trả về một số ngẫu nhiên của single
   Next Count
End Sub
```

Trong ví dụ trên vòng lặp sẽ được thực hiện từ Count=1 đến Count=100. Vòng lặp sẽ điền một giá trị ngẫu nhiên (hàm **Rnd**) vào các ô so với ô hiện tại bằng cách dùng phương thức **Offset**. Trong ví dụ này bước (**step**) với giá trị mặc định là 1. Tức là count bắt đầu bằng 1 và sau đó giá trị count sẽ là 2,3,4,5... cho tới khi 100.

```
Sub FillRange()
   Dim Count As Integer
   For Count = 1 To 100 Step 2
      ActiveCell.Offset(Count - 1, 0) = Rnd
      'Rnd hàm trả về một số ngẫu nhiên của single
   Next Count
End Sub
```

Cũng tương tự như ví dụ FillRange đầu nhưng có đưa bước thực hiện vào. Tức là vòng lặp bắt đầu bằng 1, các giá trị sau lần lượt là 3,5,7,... và giá trị cuối cùng là 99. Giá trị bước (**step**) định nghĩa biến đếm sẽ được tăng như thế nào.
Vậy ví dụ về bad loop ở trên bạn có thể sửa lại như sau:

```
Sub BadLoop()
   Dim StartVal As Integer
   Dim NumToFill As Long
   Dim CellCount As Long
   StartVal = CInt(InputBox("Xin nhập vào giá trị bắt đầu: "))
   NumToFill = CInt(InputBox("Điền bao nhiêu ô? "))
   ActiveCell = StartVal
   For CellCount = 1 To NumToFill
      ActiveCell.Offset(CellCount-1, 0) = StartVal + CellCount - 1
   Next CellCount
End Sub
```

Chú ý, vòng lặp Next-For có thể lồng vào nhau. Các bạn hãy xem ví dụ sau

```
Sub FillRange2()
   Dim Col As Integer
   Dim Row As Long
   For Col = 1 To 5
      For Row = 1 To 12
         Cells(Row, Col) = Rnd
      Next Row
   Next Col
End Sub
```

E. Cấu trúc **FOR EACH …**
Cấu trúc vòng lặp như sau:

```
For Each element In collection
   [statements] (các phát biểu)
   [Exit For] (thoát khỏi vòng lặp For)
   [statements] (các phát biểu)
Next [element]
```

Xem ví dụ sau:

```
Sub DeleteRow1()
   Dim WkSht As Worksheet
   For Each WkSht In ActiveWorkbook.Worksheets
      WkSht.Rows(1).Delete
   Next WkSht
End Sub
```

Trong ví dụ này biến WkSht là biến đối tượng đại diện cho mỗi worksheet trong ActiveWorkbook.
Ví dụ sau sẽ quét qua tất cả các **Cell** trong một **Range**.

```
Sub ChangeSign()
   Dim Cell As Range
   For Each Cell In Range("A1:E50")
      If isnumberic(Cell.Value) Then
         Cell.Value = Cell.Value * (-1)
      End If
   Next Cell
End Sub
```

CHƯƠNG III. MỘT SỐ LỆNH THÔNG DỤNG

1 .EXIT FOR

Câu lệnh : EXIT FOR
Lồng vào trong vòng lặp For khi muốn dừng lại vòng lặp bất cứ lúc nào.

2 .EXIT DO

Câu lệnh : EXIT DO
Lồng vào trong vòng lặp có cấu trúc DO khi muốn dừng lại vòng lặp bất cứ lúc nào.

3 .EXIT SUB

Câu lệnh : EXIT SUB
Thoát khỏi thủ tục mà bất cứ lúc nào mà không cần thực hiện các lệnh bệnh trong nó.

4 .END

Chấm dứt chương trình ngay, tất cả các cửa sổ chương trình đều đóng lại khi bạn thực hiện thao tác này.

5. Beep

Phát ra tiếng kêu Beep

6. Lệnh Date :

Cho phép bạn đặt lại ngày hệ thống, hay lấy ngày hệ thống
Cú pháp : **DATE** = <ngày bạn đặt>
VD: Date = #June 12, 2000#

7. TIME

Cho phép đặt lại giờ hệ thống, hay lấy giờ hệ thống
Cú pháp : **TIME** = <Giờ bạn đặt>
VD: Time = # 5 : 12 : 45 PM #

8.LOAD

Nạp 1 form (dùng nó để mở 1 Form)
Cú pháp : **LOAD** <Tên Form>
Để làm xuất hiện hoặc ẩn đi sử dụng phương thức **SHOW**, ví dụ form1.**Show** hay form.**Hide**

9. Lệnh ChDrive

Dùng để đổi ổ đĩa làm việc
Cú pháp : **ChDrive** <"Tên ổ đĩa :">

10. MkDir

Dùng để tạo một thư mục mới trên đĩa

Cú Pháp : MkDir <Đường dẫn>
Ví dụ : MkDir “D:\ChuyendeVBA”

11. Lệnh ChDir
Lệnh này dùng để thay đổi thư mục làm việc tại ổ đĩa đang làm việc
Cú pháp : ChDir <”Đường dẫn thư mục”>

12. Lệnh RmDir :
Dùng để xóa 1 thư mục rỗng.
Cú pháp : RmDir <”Đường dẫn thư mục”>

13. Lệnh KILL
Xóa 1 hay nhiều tập tin trên đĩa
Cú pháp : KILL <”Đường dẫn đền tập tin”>
Ví dụ : Kill “D:\baitapVBA.txt”
Kill “D:*.txt”

14. Lệnh NAME :
Dùng để đổi tên tập tin
Cú pháp : NAME <”Đường dẫn tập tin cần đổi tên”> AS <”Đường dẫn và tên tệp mới”>
Ví dụ : NAME “C:\BTAP.txt” AS “C:\BAITAPVBA.txt”

15. Lệnh AppActive
Dùng để kích hoạt một cửa sổ của một chương trinhg đang chạy trên Windows
Cú pháp : **AppActive** title [Wait]
Wait : Nếu là **False** thì chương trình sẽ kích hoạt ngay khi thực hiện lệnh gọi này (VB ngầm hiểu là False).
Ví dụ : AppActive “Microsoft Word”

16. CurDir (Drive) ' trả về một chuỗi với tên đầy đủ của ổ đĩa hiện hành nếu tham số Drive để trống ( )

VD:
CurDir ( ) ---> tên ổ đĩa hiện hành
CurDir ("D") ---> tên đầy đủ "D:\"
CurDir ("X") ---> sẽ báo lỗi nếu hệ thống không có tới ổ thứ X này

17. Dir (PathName, Attributes) ' tìm kiếm tập tin với đường dẫn đặt tại tham số 1. Tham số 2(Attributs) tùy chọn để chuyên biệt thuộc tính tìm kiếm. Mặc định là vbNormal

VD:
tmp = Dir ("C:\boot.ini") ---> tmp = ""
tmp = Dir ("C:\boot.ini", vbHidden) ---> tmp = "boot.ini"

18. FileCopy (Source as String, Destination as String) ' sao chép tập tin từ đường dẫn nguồn (Source) đến đường dẫn khác (Destination)

VD:
Filecopy "C:\Config.sys", "D:\Config.sys"

19. Filelen (PathNam As String) As Long ' trả về kích thước của tập tin

20. FileDateTime (PathName as string) ' trả về ngày tháng và thời gian tập tin đã được tạo ra hoặc được chỉnh sữa lần gần nhất.

VD:
Msgbox FileDateTime ("C:\Config.sys")

21. GetAttr (PathName as String) as Integer ' trả về một số nguyên là trị thuộc tính của File

Các hằng thuộc tính gồm:
vbNormal = 0
vbReadOnly = 1
vbHiden = 2
vbSystem = 4
vbVolume = 8
vbDirectory = 16
vbArchive = 32

VD:
GetAttr "C:\boot.ini" = 35 (file Boot.ini mang các thuộc tính sau: ReadOnly(1) + Hiden(2) + Archive(32))

22. SetAttr (PathName as String, Attributes as vbFileAttribute) ' Xác lập thông tin thuộc tính của File. Sử dụng các hằng thuộc tính hoặc các giá trị ở bảng trên cho tham số Attributes
VD:
SetAttr "C:\Boot.ini, 0" ---> xác lập file boot.ini chỉ mang một thuộc tính là Normal
SetAttr "C:\Boot.ini,3" ---> xác lập boot.ini mang thuộc tính chỉ đọc và ẩn.

23. FreeFile ' trả về một số integer là chỉ số (ID) để HDH theo dõi và quản lý các file đang mở (Open). Dùng một biến nguyên để lưu giá trị này dùng cho các cuôc gọi lệnh Open (file). Khi đó ta không cần quan tâm đến các chỉ số(ID) này nữa. FreeFile sẽ theo dõi và cung cấp cho ta các chỉ số(ID) chưa dùng.

VD:
Dim Filenum#
FileNum = FreeFile
Open [PathName] For Output As #Filenum

24. Open [PathName as String] For [Mode] As [ID File] ' Mỗi khi thấy lệnh Open, VB sẽ sẵn sàng cho các thao tác đọc và ghi lên File được cung cấp ở tham số [PathName]. Có năm chế độ mở tập tin được đặt ở tham số [Mode], và [ID File] dĩ nhiên là chỉ số của tập tin đang mở.
Các dạng thức cơ bản của lệnh Open:
Open [Pathname] For Input As Filenum ' mở File và chỉ đọc được thông tin, không ghi lên được.
Open [PathName] For Output As Filenum ' mở File để xuất thông tin. Khi được mở theo dạng này mọi thông tin cũ trên File sẽ bị mất.
Open [PathName] For Append As Filenum ' mở File để đọc và ghi tiếp lên được.
Open [PathName] For Random As Filenum ' mở và truy cập ngẫu nhiên các bản ghi và các trường trên File (phải biết được cấu trúc của các bản ghi)
Open [PathName] For Binary As Filenum ' đọc ghi theo Byte. Đây là dạng tổng quát và linh hoạt nhất.

25. Input [Number, #Filenum] ' đọc nội dung File với số lượng xác định ở tham số Number

VD: Str = Input (10, #Filenum) ' đọc 10 ký tự vào biến Str.

Dạng khác của Input:

Input [#Filenum, Str] ' ở đây biến Str thường ở dạng Variant

Line Input [#FileNum, Str] ' đọc thông tin theo từng dòng vào biến Str. Thường kết hợp với hàm EOF để lấy hết thông tin

26. EOF (Filenum) ' trả về vị trí chấm dứt của File khi đang mở

VD:

Do While Not EOF (Filenum) ' lập nếu không phải ở end of file

Line Input #Filenum, Str ' đọc từng dòng vào biến Str

Loop

27. LOF (Filenum) ' trả về kích thước của File khi đang mở.

VD:

Str = Input (LOF(Filenum), #Filenum) ' sẽ lấy hết nội dung của File (không nên sử dụng với File có kích thước vài Mb sẽ bị lỗi "out of memory")

28. Write [#Filenum, Expression] ' ghi lên File

VD:

Write #FileNum, "12345" ---> "12345"

Write #Filenum, 12345 --->12345

29. Print [#Filenum, Expression] ' làm việc chính xác như khi Print lên Form

VD:

Print #Filenum, 123; 456 ---> 123 456

30. Get [#Filenum, position, ByteArray] ' lấy thông tin từ tập tin được mở theo Binary tại vị trí xác định bởi Position và lưu vào ByteArray. Số byte lấy ra tùy thuộc vào kích thước của mảng ByteArray. Mỗi khi lấy ra 1 byte con trỏ tập tin tự động chuyển tới vị trí byte kế tiếp.

VD:

Dim Str as String * 4

Get #Filenum, 3, Str ---> lấy 4 byte bắt đầu từ byte thứ 3 lưu vào Str.

(nếu có câu lệnh Get tiếp theo mà tham số Position bỏ trống, thì vị trí bắt đầu lấy ra sẽ là byte thứ 8)

31. Loc (#Filenum) ' Trả về vị trí byte đọc/ghi hiện tại trong tập tin đang mở.

32. Seek [#Filenum, Position] ' dịch chuyển con trỏ tập tin đến vị trí qui định bởi tham số Position

VD:

Seek #Filenum, 3

Get #Filenum, , Str ---> vị trí lấy ra sẽ bắt đầu tại byte thứ 3

33. Seek (#Filenum) ' trả về vị trí hiện tại của con trỏ tập tin

34. Put [#Filenum, Position, ByteArray] ' đặt nội dung của mảng ByteArray vào vị trí

byte thứ[Position].
Lệnh Put se ghi đè lên mọi thứ và chỉ dùng cho truy cập Random và Binary
VD: Put #filenum, , Str ---> sẽ ghi 4 byte bắt đầu ở vị trí byte thứ 1

35. Reset ' Đóng tất cả các tập tin đã được mở bằng lệnh Open

```
1. Option Explicit
2. Private Sub Form_Load()
3. Dim FileNumber
4.     For FileNumber = 1 To 5
5.         Open "TEST" & FileNumber For Output As
   #FileNumber ' Mở file
6.             Write #FileNumber, "Hello World"   ' Ghi
   dữ liệu vào file.
7.     Next FileNumber
8.     Reset   ' Đóng file và cập nhật dữ liệu vào file
9. End Sub
```

36. Close (#Filenum) ' Đóng tập tin đã được mở bằng lệnh Open.

37. Lock [#Filenum, Expression] ' khoá tập tin không cho người khác truy cập khi App của bạn đang mở. Tham số thứ hai chuyên biệt vị trí khoá. Nếu bỏ qua tham số tùy chọn này, lệnh Lock sẽ khoá toàn bộ tập tin. Đối với các tập tin mở theo truy cập tuần tự lệnh Lock sẽ khoá toàn bộ tập tin bất kể khoảng do tham số 2 qui định.

VD:
Lock #Filenum, 1 To 100 ---> sẽ khoá 100 byte từ byte thứ 1

38. Unlock [#Filenum, Expression] ' mở khoá tập tin, tham số sử dụng như Lock.

Chú ý: cần bảo đảm loại bỏ tất cả các khoá với câu lệnh Unlock tương ứng trước khi đóng tập tin hoặc thoát khỏi chương trình(các đối số phải tương hợp chính xác). Nếu không tập tin có thể bị rối loạn.

39. Open:
Để điều khiển sự chia sẽ tập tin vào thời gian bạn mở tập tin. Có thể dùng câu lệnh tổng quát nhất của lệnh Open
Cú pháp:
Open PathName [For mode] [Access access] [Lock] As #Filenum [Len=reclength] '
trong đó:

PathName là chuỗi chứa đường dẫn đến tập tin
Mode là từ khoá chuyên biệt chế độ tập tin như Input, Append, Random...
Access là từ khoá chuyên biệt các thao tác được phép trên tập tin mở. Có ba thao tác: Read, Write, ReadWrite
VD:Open PathName For Binary Access Read As #Filenum --> cho phép bạn đọc nhưng không cho phép thực hiện các thay đổi đối với tập tin.

Lock là từ khoá chuyên biệt các thao tác được phép trên tập tin mở đối với các quá trình khác.
Khác với Access : điều khiển cách thức chương trình bạn làm việc với tập tin. Với từ khoá Lock có bốn khả năng:

1- [Shared] các quá trình khác có thể đọc và viết vào tập tin mặc dù chương trình của bạn đang làm việc với tập tin đó.
2- [LockRead] chương trình khác không thể mở để đọc tập tin, trong khi chương trình của bạn đang làm việc với tập tin đó.
3- [LockWrite] không thể mở để viết lên tập tin trong khi chương trình bạn đang làm việc với tập tin đó.
4- [LockReadWrite] chương trình khác không thể làm việc với tập tin trong khi chương trình bạn đang làm việc với tập tin đó.

VD: Open PathName For binary Access Lock Read #Filenum ---> sẽ ngăn chặn các chương trình khác sử dụng tập tin khi bạn đang làm việc với tập tin đó.

Reclength đây là số nguyên từ 1 đến 32767. Đối các tập tin mở ở chế độ Random số này đưa ra chiều dài bản ghi. Đối với các tập tin chuỗi thứ tự, giá trị này là số lượng các ký tự đươc đệm trong hệ điều hành.

CHƯƠNG IV. MỘT SỐ HÀM THƯỜNG DÙNG

Tất cả các hàm đều có dạng : Tên hàm (các đối số)

1. Hàm Abs (Number)
Trả về một giá trị là giá trị tuyệt đối của Number

2. Hàm Sin (Number as Double)
Trả về một số thực là Sin của một góc (tình bằng đơn vị Radian)

3. Hàm Cos (Number as Double)
Trả về một số thực là Cos của một góc (tình bằng đơn vị Radian)

4. Hàm Tan (Number as Double)
Trả về một số thực là Tan của một góc (tình bằng đơn vị Radian)

5. Hàm Atn (Number as Double)
Trả về một số thực là ArcTan của một góc (tình bằng đơn vị Radian)

6. Hàm Int (Number) :
Trả về phần nguyên của Number nếu nó là số dương, còn nếu số âm thì có giá trị nhỏ hơn phần nguyên 1 đơn vị

7. Hàm Fix (Number)
Trả về phần nguyên của Number nếu nó là số dương, còn nếu số âm thì có giá trị lớn hơn phần nguyên 1 đơn vị

8. Hàm Sgn (Number)
Trả về một số nguyên
Nếu Number > 0 sẽ trả về 1
Nếu Number < 0 sẽ trả về -1
Nếu Number = 0 sẽ trả về 0

9. Hàm Sqr (Number)

Trả về căn bậc hai của Number

10. Hàm Exp (x)

Đưa ra e lũy thừa x, e là cơ số Logarit tự nhiên. Hàm trả về một số thực

11. Hàm Log (x)

Đưa ra Logarit tự nhiên của x

12. Hàm Round (Expression [số])

Hàm này sẽ làm tròn số

[,số] : số làm tròn qua chấm thập phân.
VD : Round(9.7) = 10
Round (9.785 , 2) = 9.79

13. Rnd (Number)

Tạo 1 số ngẫu nhiên là 1 số thực từ 0 đến Number, với Number là 1 số nguyên.

14. Hàm Now :

Hàm này trả về ngày tháng năm và thơi gian hiện hành.

15. Hàm Day (NgayThangNam)

Trả về ngày trong NgayThangNam mà bạn ghi.
Ta thường sử dụng Day(Now) để lấy ngày hệ thống

16. Hàm Month (NgayThangNam)

Trả về Tháng trong NgayThangNam mà bạn ghi.
Ta thường sử dụng Month(Now) để lấy tháng hệ thống

17. Hàm Year (NgayThangNam)

Trả về Năm trong NgayThangNam mà bạn ghi.
Ta thường sử dụng Year(Now) để lấy năm hệ thống

18. Hàm Weekday (NgayThangNam)

Trả về ngày thứ mấy trong tuần ứng với NgayThangNam mà bạn nhập vào
Ta có thể sử dụng Weekday(Now) để lấy thứ của ngày hiện tại

19. Hàm Hour (ThoiGian)

Trả về giờ ứng với ThoiGian mà bạn nhập vào
Ta có thể sử dụng Hour(Now) để lấy giờ của hệ thống hiện tại

20. Hàm Minute (ThoiGian)

Trả về phút ứng với ThoiGian mà bạn nhập vào
Ta có thể sử dụng Minute(Now) để lấy phút của hệ thống hiện tại

21. Hàm Second (ThoiGian)

Trả về giây ứng với ThoiGian mà bạn nhập vào
Ta có thể sử dụng Second(Now) để lấy giây của hệ thống hiện tại

22. Hàm Replace(chuoi, chuoicantim, chuoithaythe, Vitrithaythe, solanthaythe)

Hàm này sẽ trả về một chuỗi mới theo ý nghĩa như trên. Ví dụ:

Replace(“2322” , ”2” , “5” , 1 , 2) = “5352”
Replace(“2322” , ”2” , “5” , 2 , 2) = “355”

23. Hàm Val(String)

Hàm này có tác dụng đổi **1 chuỗi thành 1 số**, nếu chuỗi nãy có kí tự đầu là ký tự thì sẽ trả về 0.

24. Hàm Str (Number)

Ngược lại Hàm Val, hàm này có tác dụng **đổi 1 số thành 1 chuỗi**.

25 Hàm QBColor (color)

Sẽ cho bạn màu của một đối tượng nào đó, thể hiện từ 0 đến 15.
Ví dụ : QBColor (0) sẽ cho màu đen, QBColor (4) sẽ cho màu đỏ ,…

26. Hàm RGB (Red, Green, Blue)

Chọn một màu theo một tỉ lệ nào đó ngoài các màu từ 0 – 15. Nó sẽ là sự kết hợp của 3 màu.

27. Hàm Asc (String)

Sẽ trả về một con số, con số này là mã ASCII của kí tự String, nếu là một chuỗi gồm nhiều kí tự thì kí tự sẽ lấy kí tự đầu tiên.

27b. AscW (string) ' Chuyển ký tự thành mã Ascii (hỗ trợ Unicode)
VD: AscW(“ệ”) = 7879 = H1EC7

28. Hàm Chr(CharCode)

Hàm trả về một kí tự tương ứng với một mã ASCII nào đó.
ChrCode là mã ASCII của kí tự mà bạn cần biết

28b.ChrW(charcode) ' Chuyển mã Ascii thành ký tự (Hỗ trợ Unicode)
VD: ChrW(&H1EC7) = “ệ”

29. Hàm Len (String)

Trả về dộ dài của chuỗi String, kể cả khoảng trắng

30. Hàm Ltrim (String)

Hàm trả về chuỗi mới sau khi cắt bỏ các khoảng trắng bên trái chuỗi String

31. Hàm Rtrim (String)

Hàm trả về chuỗi mới sau khi cắt bỏ các khoảng trắng bên phải chuỗi String

32. Hàm Trim (String)

Hàm trả về chuỗi mới sau khi cắt bỏ các khoảng trắng bên trái và bên phải chuỗi String

33. Hàm Left (String,n)

Trả về một chuỗi kí tự (kể cả khoảng trắng) được cắt từ bên trái của chuỗi String, số kí tự cắt lấy là n.

34. Hàm Right (String,n)

Trả về một chuỗi kí tự (kể cả khoảng trắng) được cắt từ bên phải của chuỗi String, số kí tự cắt lấy là n.

35. Hàm MID (String, Start, [Length])

Trả về một chuỗi, chuỗi này được lấy từ chuỗi String và bắt đầu từ Start và lấy Length ký tự.

Nếu Length bỏ trống hoặc lớn hơn độ dài String thì coi như lấy từ vị trí Start cho đến hến.

36. Hàm Space (Number)

Hàm trả về một chuỗi gồm Number khoảng trắng

37. Hàm String (Number, Character)

Trả về một chuỗi gồm Number kí tự giống nhau và giống Character

38. Hàm InStr (Start, String1, String2, Compare)

Hàm này dùng để tìm một chuỗi con có nằm trong chuỗi mẹ hay không, nếu tìm thấy thì sẽ cho biết nằm ở vì trí thứ mấy của chuỗi mẹ.

Start : Tìm bắt đầu từ vị trí Start trong chuỗi mẹ, nếu không ghi thì tìm ở vị trí đầu tiên

String1: Chuỗi mẹ

String2 : Chuỗi con

Compare : có các giá trị 0, 1, 2

+ 0 : so sánh chính xác từng kí tự, đây là giá trị mặc nhiên
+ 1 : So sánh không phân biệt chữ hoa và chữ thường
+ 2 : chỉ dùng trong khi lập trình cho MS Access

* Khi dùng đến đối số Compare thì đối số Start không được bỏ trông

39. Hàm Ucase (String)

Trả về mộy chuỗi kí tự viết hoa của chuỗi String

40. Hàm Lcase (String)

Trả về một chuỗi kí tự viết thường của chuỗi String

41. Hàm Format (Value, format)

Hàm này dùng để định dạng theo ý của bạn

Value : Giá trị cần định dạng

Format : Các kí hiệu định dạng.

0 nếu có giá trị thì thể hiện giá trị đó, nếu không có thì ghi số 0. Nêu số 0 ít hơn thì giá trị vẫn được ghi đầy đủ

: Thể hiện các giá trị tương ứng, nếu kí tự số ở vị trí đó không có thì bỏ qua, nếu # ít hơn thì giá trị vẫn được ghi đầy đủ.

$: Dấu $ bạn có thể dùng chung với số 0 hay #

. : Dấu ngăn cách phần thập phân

, : Dấu ngăn cách phần nghìn

% : Khi có kí hiệu phần trăm này trong đối số Format, con số sẽ tự thêm % vào sau

dd/mm/yyyy : Định dạng Ngày Tháng Năm, với đối số Value = Now

hh:mm:ss AM/PM : Định dạng Giờ phút giây theo dạng, với đối số Value = Now

hh:mm:ss AM/PM dd/mm/yyyy : Định dạng Giờ phút giây vừa định dạng ngày tháng năm theo dạng, với đối số Value = Now

hh:mm : Định dạng chỉ có giờ và phút với Value=Now.

Ví dụ :
Format(12345.5 , “0000000.00”) = “012345.50”
Format(12345.5, “######.##”) = “12345.5”
Format(12345.5, “$###.##”) = “$12345.5”
Format(0.34, “###%”) = “34%”

42. Hàm IIF(<Điều kiện>, Truepart, Falsepart)

Hàm này sẽ trả về giá trị true nếu điều kiện đúng và cho False khi điều kiện sai.
Hàm này là cách viết ngắn họn của IF…END IF

Ví dụ txt1.text = IFF(x<500,”Lương bạn còn thấp”,”Bạn đã có lương cao”)

43. InStr (start, string1, string2, compare) ' trả về vị trí bắt đầu của một chuỗi con cần tìm trong một chuỗi mẹ. tham số 1(start) xác định vị trí bắt đầu tìm, tham sô 2(string1) là chuỗi mẹ, tham số 3(string2) là chuỗi cần tìm, tham số 4(compare) mặc định là so sánh nhạy ký tự.
Khi bỏ qua tham số thứ nhất thì vị trí bắt đầu tìm mặc định là 1

VD: *pos = InStr ("caulacboVB", "VB")* ---> pos = 9

44. InStrRev (StringCheck as string, StringMatch as string, Start as Long, Compare) ' chức năng như InStr nhưng InStrRev hoạt động ngược lại từ cuối chuỗi và cú pháp khác hơn. Cả hai hàm đều là hàm tìm kiếm nhạy ký tự nên cần chú ý chữ thường và chữ HOA. InStrRev thường kết hợp với Mid để tách một tên File khỏi đường dẫn và tên mở rộng.

VD:

PathFile = "C:\temp\001.tmp"
Pos = InStrRev (pathFile, "\")
PathFile = Mid (PathFile, Pos + 1)
Pos = InStrRev (PathFile, ".")
PathFile = Mid (PathFile, 1, Pos - 1) --->PathFile = "001"

45. StrComp (String1, String2, Compare) ' dùng để so sánh 2 chuỗi.
Trị trả về: (String1 < String2) = -1; (String1 = String2) = 0; (String1 > String2) = 1

46. Like ' so sánh 2 chuỗi cho phép sử dụng biệt ngữ (như dùng ký tự đại diện trong Dos) trị trả về = True nếu tương hợp

VD: "abcd" Like "*bcd" = True
"abcd" Like "a?cd" = True
"a1cd" Like "a#cd" = True
Chú ý hàm Like mặc định cũng là hàm nhạy ký tự, theo thiết lập Option Compare ở form hoặc module

47. Join (SourceArray, Delimiter) ' tạo chuỗi mới từ một mảng chuỗi (SourceArray) với các phần tử được phân định bởi tham số Delimiter

VD:

Arr (0) = "a"
Arr (1) = "b"
Arr (2)= "c"
Print Join (Arr, " ") = "a b c"

48. Split (Expression as String, Delimiter, Count, Compare) ' tạo mảng chuỗi từ một

chuỗi (Expression). Đặt tham số Delimiter để chuyên biệt chỗ ngắt, nếu bỏ wa tham số này mặc định Split sẽ tách tại các khoảng trống của chuỗi. Tham số Count qui định số lần tách. Ba tham số cuối là tùy chọn
VD:
Dim str as string, Arr as Variant
str = "Chuyen de VBA"
Arr = Split (str) ' dùng For duyệt mảng Arr sẽ cho ra : Arr (0) = "Chuyen"; Arr (1) = "de"; Arr (2) = "VBA"
Arr = Split (str, "e") ---> Arr (0) = "Chuy"; Arr (1) = "n d"; Arr (2) = "VBA"
Arr = Split (str, " ", 2) ---> Arr (0) = "Chuyen"; Arr(1) = "de VBA"

49. Filter (sourcearray, match [, include [, compare]]) ' Lọc mảng sourcesrray với giá trị lọc là match ; include: Lọc đảo (True hoặc False) ; compare: chỉ rõ kiểu dữ liệu để so sánh trong quá trình lọc.
Dùng cho tham số compare
vbUseCompareOption = –1 : Chế độ tùy chọn, VB sẽ tự động lựa loại dữ liệu thích hợp
vbBinaryCompare = 0 : So sánh nhị phân
vbTextCompare = 1 : So sánh chuỗi
vbDatabaseCompare = 2 : So sánh dữ liệu

VD:

```
Dim selNames() As String
Dim Names(1 To 5) As String
Names(1) = "A"
Names(2) = "B"
Names(3) = "C"
Names(4) = "D"
Names(5) = "E"
selNames = Filter(Names, "A") ' Returns "A"
selNames = Filter(Names, "B", False) 'Returns "A" , "C", "D", "E"
```

50. StrReverse(expression as String) ' Đảo chuỗi expression

VD: StrReverse("1234567") ' Returns "7654321"