TRƯỜNG CAO ĐẲNG NGHỀ CÔNG NGHIỆP HÀ NỘI

Hoàng Anh Thơ (chủ biên)

GIÁO TRÌNH

LẬP TRÌNH MACRO TRÊN OFFICE

(Lưu hành nội bộ)

Hà Nội năm 2013

LẬP TRÌNH MACRO TRÊN MS OFFICE

MỤC LỤC

Bài mở đầu: TỔNG QUAN VỀ NGÔN NGỮ VBA

Thời gian:1 giờ

1. Giới thiệu về VBA

VB (Visual Basic) do Bill Gates và Alan Cooper viết từ thời dùng cho máy tính 8 bits 8080 hay Z80. Hiện nay nó chứa đến hàng trăm câu lệnh (*commands*), hàm (*functions*) và từ khóa (*keywords*) và là một ngôn ngữ lập trình phổ biến.

"***Visual***" đề cập đến phương pháp được sử dụng để tạo giao diện đồ họa người dùng (*Graphical User Interface - GUI*), có sẵn những bộ phận hình ảnh, gọi là Controls, để sắp đặt vị trí và quyết định các đặc tính của chúng trên một khung màn hình, gọi là Form.

"***Basic***" đề cập đến ngôn ngữ BASIC (*Beginners All-Purpose Symbolic Instruction Code*), là một ngôn ngữ lập trình đơn giản, dễ học.

VBA (Visual Basic for Application) là phần mở rộng của ngôn ngữ VB, nằm phía sau các chương trình ứng dụng nền (Word, Excel, MSAccess, MSProject…) được gọi là Macro. VBA được dùng để nâng cao các tính năng của ứng dụng nền bằng cách tự động hóa các chương trình. VBA giúp người dùng có thể tạo ra các tuỳ biến mạnh hơn, thân thiện hơn với trong công việc của mình.

2. Đặc điểm của VBA

- Ngôn ngữ lập trình Visual Basic (VB) dễ sử dụng, hướng sự kiện, tuỳ theo sự kiện diễn ra với đối tượng mà lập trình theo modul, theo lớp.
- Lập trình trực quan, trong môi trường Windows, cung cấp cho người sử dụng một bộ công cụ hoàn chỉnh để đơn giản hóa việc triển khai lập trình ứng dụng.
- Giao diện lập trình là đồng nhất trên tất cả các ứng dụng nền (Word, Excel, Access, …) hỗtrợ VBA, , do đó người dùng có thể lập trình mở rộng trên nhiều ứng dụng nền một cách thuận lợi.

- Thư viện lập trình có rất nhiều và đa dạng cho nên người dùng có thể xây dựng ứng dụng của mình nhanh và chuyên nghiệp.
- Tốc độ thực thi của chương trình nhanh.
- Khai thác được hầu hết các tính năng sẵn có của ứng dụng nền.
- Chương trình VBA có thể được nhúng trong tệp của ứng dụng hoặc có thể được lưu dưới dạng một dự án độc lập. Điều này giúp cho việc phân phối, chia sẻ mã lệnh được thuận tiện.

Từ các đặc điểm cơ bản đã được phân tích ở trên có thể thấy rằng VBA là một công cụ lập trình cho phép phát triển nhanh phần mềm và được tích hợp vào trong ứng dụng nền. Về thực chất, VBA được xây dựng dựa trên kiến trúc COM *(Component Object Model: là một kiến trúc lập trình được thiết kế bởi Microsoft với mục đích tạo ra một chuẩn công nghệ trong lập trình, mà ở đó cho phép xây dựng chương trình theo mô hình lắp ghép hay sử dụng lại các sản phẩm đã được hoàn thiện từ trước theo chuẩn COM)*, cho nên người dùng có thể sử dụng các thành phần sẵn có của ứng dụng nền trong việc xây dựng chương trình của mình với VBA.

Một dự án được xây dựng bằng VBA dựa trên ứng dụng nền nào thì nó phụ thuộc chặt chẽ vào ứng dụng nền đó, bởi theo mặc định, dự án VBA sẽ hoạt động và sử dụng các thành phần trong chính ứng dụng nền đó. Điều này có nghĩa là rất khó có thể chuyển đổi một dự án VBA từ loại ứng dụng nền này sang một ứng dụng nền khác cũng như tạo ra một ứng dụng chạy độc lập.

Sự khác biệt cơ bản nhất của VBA trong các ứng dụng nền (ví dụ giữa VBA trong Word và VBA trong Excel) là cách thức sử dụng các thành phần (đối tượng) của ứng dụng nền.

3. Cấu trúc một dự án VBA

Khi nói đến các thành phần tạo nên một dự án VBA thì cấu trúc của nó, về tổng quát, như sau:

- ***Mô-đun chuẩn*** (Module): là nơi chứa các mã lệnh khai báo, các chương trình con (hàm và thủtục). Việc tạo ra các mô-đun chuẩn

thường căn cứ theo các khối chức năng mà người thiết kế hệ thống đặt ra.

- ***Mô-đun lớp*** (Class Module): là nơi chứa định nghĩa cho các lớp của dự án.
- ***UserForm***: là giao diện giúp cho việc giao tiếp giữa người sử dụng và chương trình được thuận tiện. Thông thường người ta sử dụng Userform để nhập số liệu, xuất kết quả của chương trình. Trong một số dự án, nếu việc nhập số liệu và biểu diễn kết quả được thực hiện trực tiếp trên ứng dụng nền (Word, Excel, Access…), thì có thể không cần sử dụng Userform.

Những thành phần trên là bộ khung để người dùng xây dựng chương trình của mình lên trên đó, ví dụ như viết mã lệnh hay thiết kế giao diện cho chương trình. Mô-đun lớp và UserForm là hai thành phần có thể xuất hiện hoặc không tùy thuộc vào từng dự án và tất cả những thành phần sử dụng trong dự án đều được hiển thị trên giao diện của VBA IDE.

Tuy nhiên, khi xây dựng chương trình (viết mã lệnh) cụ thể thì khái niệm cấu trúc của một chương trình là sự bốtrí, sắp xếp các câu lệnh trong chương trình đó. Đối với ngôn ngữ lập trình Visual Basic (VB), cấu trúc của nó chỉ tập trung vào chương trình con (hàm và thủtục) chứ không có một quy định về cấu trúc nào đối với chương trình chính.

4. Môi trường phát triển tích hợp VBA IDE

Trong mỗi công cụ lập trình trên ứng dụng luôn có một môi trường lập trình nhằm hỗ trợ người dùng có thể xây dựng, thử nghiệm và hoàn thiện chương trình của mình. Trong Word và Excel, khi sử dụng VBA để lập trình, môi trường lập trình được gọi là Môi trường phát triển tích hợp (viết tắt là VBA IDE - *Integrated Development Environment*)

IDE của Visual Basic là nơi tập trung các menu, thanh công cụ và cửa sổ để tạo ra chương trình. Mỗi một thành phần của IDE có các tính năng ảnh hưởng đến các hoạt động lập trình khác nhau.

a. Menu Bar (Thanh thực đơn)

Chứa đầy đủ các menu commands sử dụng để làm việc với VB6, kể cả các menu để truy cập các chức năng đặc biệt dành cho việc lập trình chẳng hạn như Project, Format, hoặc Debug.

b. Toolbars (Thanh công cụ)

Các toolbars có hình các icons cho phép click để thực hiện công việc tương đương với dùng một menu command, nhưng nhanh và tiện hơn.

- Mở (đóng) một toolbar: ***View → Toolbars*** (Debug, Edit, form Editor, Standard)

- Sửa đổi các toolbars bằng lệnh: ***View →Toolbars → Customize...***

c. Toolbox (Hộp công cụ)

Toolbox là hộp chứa các các điều khiển, gọi là controls, chúng có thể được đặt lên các giao diện (user form) trong lúc thiết kế (design).

- Mở hộp toolbox: ***View → Toolbox***.

- Thêm các điều khiển controls vào toolbox: ***Project → Components***. Hoặc kích chuột phải vào toolbox → chọn Componenst.

d. Project Explorer Window (Cửa sổ dự án)

Liệt kê các forms và các modules trong project hiện hành. Một project là sự tập hợp các files sử dụng để tạo một trình ứng dụng.

e. Properties window (Cửa sổ thuộc tính)

Liệt kê các thuộc tính của forms hoặc các điều khiển (controls) được chọn, chẳng hạn như size, caption, color, font....

g. Form Designer (Giao diện thiết kế)

Dùng để thiết kế giao diện lập trình, bổ sung các điều khiển (controls), các đồ họa (graphics), các hình ảnh.

h. Code Window (Cửa sổ mã lệnh)

Tại cửa sổ này hiển thị tất cả các sự kiện (thủ tục) đã có của các điều khiển trên Form hiện tại.

Bài 1: CƠ BẢN VỀ NGÔN NGỮ LẬP TRÌNH VB

Thời gian:13giờ

1. Những qui định về cú pháp

Cú pháp được hiểu là một một tập hợp bao gồm các quy tắc, luật lệ về trật tự và hình thức viết của một câu lệnh hay một cấu trúc lệnh.

Trong ngôn ngữ lập trình Visual Basic (VB), cũng như các ngôn ngữ lập trình khác, đều có những quy định về cú pháp cho việc viết mã lệnh và người lập trình cần phải tuân theo các quy tắc này để trình biên dịch có thể dịch mã lệnh mà không phát sinh lỗi. Sau đây là các quy định cơ bản về cú pháp của VB:

- Các câu lệnh phải là các dòng riêng biệt. Nếu có nhiều lệnh trên cùng một dòng thì giữa các lệnh ngăn cách nhau bằng dấu hai chấm (:). Nếu dòng lệnh quá dài, muốn ngắt lệnh thành hai dòng thì sử dựng dấu cách và dấu gạch dưới (_).
- Nếu muốn chèn thêm ghi chú, phải bắt đầu dòng chú thích bằng dấu nháy đơn (').
- Qui ước khi đặt tên:
    - phải bắt đầu bằng kí tự kiểu chữ cái thông thường;
    - không chứa dấu chấm, dấu cách hay các ký tự đặc biệt khác;
    - tên thủ tục, biến, hằng không quá 255 ký tự, tên của điều khiển, biểu mẫu, lớp và module không quá 40 ký tự;
    - không trùng với các từ khoá;
    - các đối tượng có cùng một phạm vi thì không được đặt tên trùng nhau.

2. Các trợ giúp về cú pháp trong quá trình viết mã lệnh

Các quy tắc về cú pháp thường khó, vì vậy VBA IDE cung cấp tính năng tự động phát hiện lỗi cú pháp trong quá trình viết mã lệnh.

Kích hoạt tính năng này bằng cách vào thực đơn ***Tools → Options → Editor →*** xuất hiện hộp thoại, trong mục ***Code Settings*** lựa chọn:

☑ ***Auto syntax check***: tự động kiểm tra cú pháp lệnh. *Tùy chọn này cho phép VBA IDE tự động phát hiện lỗi cú pháp ngay sau khi người dùng kết thúc dòng lệnh (xuống dòng mới). Nếu dòng lệnh lỗi cú pháp thì một hộp thoại sẽ thông báo vị trí gây lỗi cũng như nguyên nhân gây lỗi.*

☑ ***Require Variable declaration***: yêu cầu phải khai báo biến trước khi sử dụng. *Trong VB, người dùng có thể sử dụng một biến mà không cần khai báo. Trong trường hợp này biến sẽ được khởi tạo và nhận một giá trịmặc định. Tuy nhiên, nếu lạm dụng điều này, rất có thể sẽ làm cho chương trình khó quản lý và dễ nhầm lẫn, vì thếVBA IDE cung cấp tùy chọn này để cho phép người dùng thiết lập tính năng kiểm soát quá trình khai báo biến. Khi tùy chọn này được kích hoạt, tất cả các biến đều phải khai báo trước khi sửdụng và VBA IDE sẽ tự động thêm vào đầu của mỗi mô-đun dòng lệnh "Option Explicit".*

3. Tính năng gợi nhớ và tự hoàn thiện mã lệnh

Mã lệnh, thông thường là một tập hợp bao gồm các từ khóa, câu lệnh, tên biến hay toán tử được sắp xếp theo một trật tự nhất định. Tên của các thành phần này có thể khó nhớ chính xác hoặc quá dài, cho nên VBA IDE đưa ra tính năng này bằng cách hiển thị những thành phần có thể phù hợp với vị trí dòng lệnh đang soạn thảo trong một danh sách và sẽ tự động điền vào chương trình theo lựa chọn của người dùng (bấm phím Tab hoặc phím Space).

Kích hoạt tính năng này bằng cách vào thực đơn ***Tools → Options → Editor →*** xuất hiện hộp thoại, trong mục ***Code Settings*** lựa chọn:

☑ ***Auto list members***: tự động hiển thịdanh sách các thành phần của đối tượng. *Với tùy chọn này, khi một đối tượng của ứng dụng nền hay của chương trình được gọi ra để sử dụng thì một danh sách các thành phần của nó (bao gồm các phương thức và thuộc tính) sẽ được tự động hiển thị đểngười dùng chọn.*

☑ ***Auto quick infor***: tự động hiển thị cú pháp cho chương trình con (hàm hoặc thủ tục). *Với tùy chọn này, VBA IDE sẽ hiển thị những thông tin về tham*

số của một hàm hay thủ tục (đã được xây dựng từ trước) khi người dùng sửdụng nó. Các thông tin này bao gồm tên của tham số cùng với kiểu của nó.

☑ ***Auto data tips***: tự động hiển thị giá trịcủa biến. *Với tùy chọn này, trong chế độ gỡ rối (Break mode), giá trị của biến (được gán trong quá trình chạy của chương trình) sẽ được hiển thị khi người dùng đặt chuột tại vị trí biến.*

4. Từ khoá trong VB

Từ khoá là tập hợp các từ cấu thành một ngôn ngữ lập trình và mỗi ngôn ngữ lập trình đều có một bộ từ khoá riêng. Các từ khóa được dùng riêng cho những chức năng khác nhau trong ngôn ngữ lập trình, do đó việc đặt tên (hằng, biến, chương trình con ...) bắt buộc phải khác so với các từ khóa, nếu không sẽ phát sinh lỗi cú pháp.

Dưới đây là danh sách các từ khoá sử dụng trong ngôn ngữ lập trình VB:

As	For	Mid	Print	String
Binary	Friend	New	Private	Then
ByRef	Get	Next	Property	Time
ByVal	Input	Nothing	Public	To
Date	Is Null	Resume	True	WithEvents
Else	Len	On	Seek	
Empty	Let	Option	Set	
Error	Lock	Optional	Static	
False	Me	ParamArray	Step	

Chú ý: *Từ khóa trong VB không phân biệt chữ hoa, chữ thường.*

5. Các kiểu dữ liệu cơ bản

Kiểu dữ liệu là một tập hợp các giá trị mà một biến có thể nhận và một tập hợp các phép toán có thể áp dụng trên các giá trị đó. Khi một biến được khai báo thì buộc phải gán cho nó một kiểu dữ liệu nhất định.

a. Dữ liệu kiểu số

- **Byte**: kiểu số nguyên có giá trị từ 0 → 255
- **Integer**: kiểu số nguyên có giá trị từ -32768 → 32767 *(Kiểu Integer tốn ít vùng nhớ hơn các kiểu khác, nó thường dùng làm biến đếm trong các vòng lặp)*

- **Long**: kiểu số nguyên có giá trị từ -2147483648 → 2147483647 *(thường được gọi là số nguyên dài)*
- **Single**: kiểu số thực có giá trị khoảng từ -3.403*1038 → 3.403*1038 *(được gọi là độ chính xác đơn)*
- **Double**: kiểu số thực có giá trị khoảng từ 1.7977*10308 → 1.7977*10308 *(được gọi là độ chính xác kép)*

b. Dữ liệu kiểu chuỗi

Chuỗi là một hàng bao gồm các ký tự liên tục nhau, các ký tự có thể là chữ số, chữ cái, dấu cách (space), ký hiệu. Mặc định các ký tự hiển thị hoặc được nhập vào các điều khiển trên giao diện là dữ liệu kiểu chuỗi.

Có hai đặc tả chuỗi ký tự theo cú pháp như sau:

- **String * <n>**: xác định một chuỗi ký tự có độ dài cố định là n ký tự. *Trong trường hợp giá trị thực của chuỗi có độ dài ngắn hơn độ dài khai báo thì độ dài của chuỗi thì một số khoảng trắng được thêm vào cho đủ độ dài thực. Trong trường hợp giá trị thực của chuỗi có độ dài lớn hơn độ dài khai báo thì sẽ cắt bớt các ký tự dư thừa bên phải. Một chuỗi không có ký tự (độ dài bằng 0) gọi là chuỗi rỗng.*
- **String**: không xác định chiều dài tối đa của chuỗi. *Biến hay tham số kiểu chuỗi có chiều dài thay đổi, tăng hoặc giảm tùy theo dữ liệu được gán, tuy nhiên tối đa là 65.500 ký tự.*

c. Dữ liệu kiểu thời gian

Dữ liệu kiểu thời gian được khai báo bằng từ khóa: **Date**, dùng để lưu trữ và thao tác trên các giá trị thời gian (ngày và giờ).

Để cho VB biết dữ liệu là kiểu Date cần đặt dữ liệu đó giữa hai dấu thăng # (hoặc dấu nháy kép “”). *(Định dạng ngày và giờ phụ thuộc vào các thiết lập về hiển thị trong hệ thống của người dùng, là ‘dd/mm/yy’ hay ‘mm/dd/yy’)*

d. Dữ liệu kiểu logic

Dữ liệu kiểu logic chỉ chứa hai giá trị là True/False. Dữ liệu kiểu này thường dùng trong các phép toán kiểm tra.

Khai báo dữ liệu kiểu logic bằng từ khóa: **Boolean**

6. Khai báo biến trong VB

a. Biến và phạm vi của biến

Biến (Variable) là thành phần của một ngôn ngữ lập trình, giúp xử lý dữ liệu một cách linh hoạt và mềm dẻo. Trước khi sử dụng, mỗi biến sẽ được gán một kiểu dữ liệu và phạm vi sử dụng của nó gọi là khai báo biến.

Khai báo biến, về thực chất, chính là việc tạo mã lệnh (lập trình), cho nên các đoạn mã lệnh khai báo biến có thể đặt ở bất cứ thành phần nào trong dự án VBA (mô-đun chuẩn, mô-đun lớp, và Userform). Tùy theo nhu cầu sử dụng biến mà người ta giới hạn phạm vi sử dụng của biến đó sao cho việc lập trình được thuận tiện nhất.

Dựa theo phạm vi sử dụng để phân loại biến như sau:

- ***Biến cục bộ***: được khai báo trong một chương trình con, một Form hoặc một Module, sau từ khoá Dim hoặc Private. Khi đó nó chỉ có tác dụng trong một chương trình con, hoặc cục bộ trong một Form, một Module được khai báo.
- ***Biến toàn cục***: khai báo tại vị trí đầu chương trình, sau cụm từ khoá Public. Nó có tác dụng trong toàn bộ chương trình *(ở bất kỳ chỗ nào có thể viết lệnh)*.

Chú ý*: Không sử dụng các từ khoá Public hay Private cho khai báo dữ liệu nằm bên trong chương trình con (trong chương trình con sử dụng từ khóa Dim).*

b. Khai báo biến

Cú pháp:

[Public| Private| Dim] <tên biến> [As <kiểu dữ liệu>]

Cách đơn giản: Dim <tên biến> [As <kiểu dữ liệu>]

Trong đó:

- <*tên biến*>: tuân theo quy tắc đặt tên..
- [Public]: khai báo biến toàn cục
- [Private]: khai báo biến cục bộ

Chú ý:

- *Giá trị của biến luôn được thay đổi trong cả quá trình thực hiện chạy chương trình.*
- *Có thể khai báo nhiều biến trên một dòng, các biến đó phân cách nhau bởi dấu phẩy (,).*

Ví dụ:

```
Dim x
Dim a, b, c As Integer
Dim hoten As String, ngaysinh As Date
```

c. Khai báo hằng

Hằng số (Constan) là một biến đặc biệt có giá trị dữ liệu xác định, không thay đổi trong suốt quá trình chạy chương trình.

Cú pháp:

[Public|Private| Const] <tên hằng> [As <kiểu dữ liệu>] = <giá trị>

Cách đơn giản: Const <tên hằng> = <giá trị>

Trong đó:

- <*tên hằng*>: tuân theo quy tắc đặt tên.
- [Public]: khai báo hằng toàn cục
- [Private]: khai báo hằng cục bộ

Ví dụ:

```
Dim Pi = 3.14
Dim OK = True
```

7. Các toán tử và hàm thông dụng

a. Các toán tử

Toán tử (hay còn gọi là phép toán-Operator): là từ hay ký hiệu nhằm thực hiện phép tính và xử lý dữ liệu.

Có các loại toán tử sau

- Toán tử số học: (+, -, *, /, \, Mod, ^) thao tác trên các giá trị có kiểu dữ liệu số.
- Toán tử so sánh: (<, >, <=, >=, =, <>) trả về giá trị kiểu Logic

- Toán tử Logic: (And, Or, Not) thao tác trên dữ liệu kiểu Logic

b. Phép gán

Dùng để gán giá trị cho biến, hằng hoặc thuộc tính.

- Cú pháp:

<tên biến> = <giá trị>

- Ví dụ:

VB = “Visual Basic”

T1.size = 20

c. Các hàm toán học

Các hàm toán học được chứa trong thư viện Math *(có thể tra cứu thư viện này bằng Object Browser)* và có nhiệm vụ thực hiện các phép toán thông thường hay gặp. Sau đây là một số hàm thông dụng:

- Abs(N): trả về giá trị tuyệt đối của số N
- Int(N): trả về giá trị phần nguyên của số N
- Sqr(N): tính căn bậc hai của số N
- Round(N,m): làm tròn số N với m số phần thập phân
- IsNumeric(N): trả về giá trị logic là True hoặc False cho biết giá trị N có phải là kiểu số không.

d. Các hàm xử lý chuỗi

Các hàm loại này được chứa trong thư viện Strings *(có thể tra cứu thư viện này bằng Object Browser)*. Sau đây là một số hàm thông dụng:

- Trim(X): cắt bỏ các ký tự trắng ở hai đầu chuỗi X.
- Len(X): trả về độ dài của chuỗi X.
- Left(X,n): trích từ chuỗi gốc X lấy n ký tự bên trái.
- Right(X,n): trích từ chuỗi gốc X lấy n ký tự bên phải.
- Mid(X,m,n): trích từ chuỗi gốc X lấy n ký tự tính từ ký tự thứ m
- InStr(n,X,Y): trả về vị trí chuỗi con Y trong chuỗi gốc X, tìm từ vị trí n sang phải.

- InStrRev(X,Y,n): trả về vị trí chuỗi con Y trong chuỗi gốc X, tìm từ vị trí n sang trái.
- Replace(X,y,z): trả thay thế chuỗi y bằng chuỗi z trong X.
- StrReverse(X): đảo ngược các ký tự trong chuỗi X
- UCase(X): đổi tất cả các ký tự trong chuỗi X thành ký tự hoa
- Lcase(X): đổi tất các ký tự trong chuỗi X thành ký tự thường

e. Các hàm chuyển đổi kiểu dữ liệu

Các hàm loại này được chứa trong thư viện Conversion *(có thể tra cứu thư viện này bằng Object Browser)*. Sau đây là một số hàm thông dụng:

- CBool(biểu thức): chuyển thành dữ liệu kiểu logic.
- CByte(biểu thức): chuyển thành dữ liệu kiểu Byte.
- CDbl(biểu thức): chuyển thành dữ liệu kiểu Double.
- CInt(biểu thức): chuyển thành dữ liệu kiểu Integer.
- CLng(biểu thức): chuyển thành dữ liệu kiểu Long.
- CSng(biểu thức): chuyển thành dữ liệu kiểu Single.
- Val(chuỗi): chuyển từ kiểu chuỗi (String) thành dữ liệu kiểu Double
- Str(số): chuyển dữ liệu kiểu số thành dữ liệu kiểu chuỗi (String)
- CStr(biểu thức): chuyển thành dữ liệu kiểu chuỗi (String)
- CDate(biểu thức): chuyển thành dữ liệu kiểu Date

8. Các cấu trúc điều khiển

8.1. Cấu trúc điều kiện

a. Dạng đơn

- Cú pháp

```
If <điều kiện> Then <lệnh>
```

Hoặc

```
If <điều kiện> Then
    <khối lệnh>
```

```
End If
```

Trong đó:

+ <điều kiện>: là biểu thức logic trả về giá trị True hoặc False.

+ <khối lệnh>: có thể là một lệnh hoặc nhiều lệnh.

- Ý nghĩa

Nếu <điều kiện> đúng (True) thì chương trình sẽ thực hiện <lệnh> hoặc <khối lệnh>, nếu sai (False) thì chương trình sẽ thoát khỏi cấu trúc lệnh này, không thực hiện gì cả. *(Như vậy lệnh chỉ được thực hiện khi điều kiện đúng)*

b. Dạng rẽ nhánh

- Cú pháp

```
If <điều kiện> Then
        <khối lệnh 1>
Else
        <khối lệnh 2>
End If
```

- Ý nghĩa

Nếu <điều kiện> đúng (True) thì chương trình sẽ thực hiện <khối lệnh 1>, nếu sai (False) thì chương trình sẽ thực hiện <khối lệnh 2>.

- Ví dụ: Viết hàm kiểm tra số chẵn hay lẻ.

```
Public Function ChanLe(So As Integer)
    If (So mod 2) = 0 then
       ChanLe = “Số chẵn”
    Else
       ChanLe = “Số lẻ”
    End If
End Function
```

c. Dạng nhiều điều kiện

- Cú pháp

```
If <điều kiện 1> Then
        <khối lệnh 1>
```

```
ElseIf <điều kiện 2> Then
        <khối lệnh 2>
… …
ElseIf <điều kiện n> Then
        <khối lệnh n>
Else
        <khối lệnh n+1>
End If
```

- Ý nghĩa

Nếu <điều kiện1> đúng thì chương trình sẽ thực hiện <khối lệnh 1>, ngược lại, nếu sai thì chương trình sẽ kiểm tra <điều kiện 2>. Nếu <điều kiện 2> đúng thì thực hiện < khối lệnh 2>, ngược lại nếu sai thì tiếp tục kiểm tra các điều kiện tiếp theo cho đến <điều kiện n> . Nếu tất cả các điều kiện kiểm tra đều sai thì chương trình thực hiện <khối lệnh n+1>. *(Như vậy điều kiện nào đúng thì thực hiện khối lệnh tương ứng, nếu tất cả đều sai thì thực hiện khối lệnh cuối cùng)*

- Ví dụ: Viết hàm kiểm tra số âm, dương, hay bằng 0.

```
Public Function Dau(So As Double)
  If (So > 0) then
        Dau = “Số dương”
  Elseif (So < 0) then
        Dau = “Số âm”
  Else
        Dau = “Số 0”
  End If
End Function
```

8.2. Cấu trúc lựa chọn

Trong trường hợp có quá nhiều các điều kiện cần phải kiểm tra, nếu ta dùng cấu trúc rẽ nhánh If…Then thì đoạn lệnh không được trong sáng, khó kiểm tra, sửa đổi khi có sai sót. Ngược lại với cấu trúc Select…Case, biểu thức điều kiện sẽ được tính toán một

lần vào đầu cấu trúc, sau đó chương trình sẽ so sánh kết quả với từng trường hợp (Case). Nếu bằng nó thi hành khối lệnh trong trường hợp (Case) đó

- Cú pháp

```
Select Case <biểu thức điều kiện>
    Case <giá trị 1>
        <khối lệnh 1>
    Case <giá trị 2 >
        <khối lệnh 2>
    … …
    Case <giá trị n>
        <khối lệnh n>
    [Case Else
        <khối lệnh n+1>]
End Select
```

Trong đó:

+ <biểu thức điều kiện>: trả về giá trị kiểu số nguyên (Byte, Integer) hoặc kí tự (String)

+ <giá trị>: có thể là một hoặc nhiều giá trị, nếu nhiều giá trị thì các giá trị này phân cách nhau bởi dấu phẩy (,)

+ <khối lệnh>: một hoặc nhiều lệnh

+ [CaseElse]: có thể có hoặc không

- Ý nghĩa

Chương trình sẽ thực hiện tính toán <biểu thức điều kiện> trước, sau đó so sánh với từng <giá trị>. Nếu kết quả của biểu thức đúng với giá trị của trường hợp nào thì thực hiện khối lệnh tương ứng của trường hợp đó. Nếu có nhiều trường hợp cùng thỏa mãn biểu thức điều kiện thì khối lệnh của trường hợp đầu tiên sẽ được thực hiện, ngược lại nếu không có trường hợp nào đúng chương trình sẽ không thực hiện khối lệnh nào cả hoặc sẽ thực hiện khối lệnh cuối cùng sau [CaseElse] nếu có.

8.3. Cấu trúc lặp xác định

Cấu trúc lặp xác định là biết trước số lần lặp bằng cách dùng biến đếm tăng dần hoặc giảm dần để xác định số lần lặp đó.

- Cú pháp

For <biến đếm> = <giá trị 1> To <giá trị 2> [Step <bước nhảy>]

<khối lệnh>

Next

Trong đó:

+ <biến đếm>: là giá trị có kiểu Integer

+ <giá trị 1>: được gán cho biến đếm, có cùng kiểu với biến đếm

+ <giá trị 2>: được so sánh với biến đếm

+ <bước nhảy>: có cùng kiểu với biến đếm, nhận giá trị âm hoặc dương. Nếu bước đếm dương thì <giá trị 1> nhỏ hơn <giá trị 2>, ngược lại nếu biến đếm âm thì <giá trị 1> lớn hơn <giá trị 2>. Khi lệnh Step không được chỉ ra thì chương trình mặc định <bước nhảy> = 1.

- Ý nghĩa

Đầu tiên chương trình sẽ gán <biến đếm> bằng <giá trị 1> sau đó thực hiện <khối lệnh>. Sau khi thực hiện <khối lệnh>, biến đếm sẽ được tăng (hoặc giảm) một đơn vị hoặc theo <bước nhảy> (nếu có lệnh Step). Cứ như vậy <khối lệnh> được thực hiện lặp đi lặp lại cho đến khi biến đếm vượt quá (lớn hơn hoặc nhỏ hơn) <giá trị 2> thì kết thúc.

- Ví dụ 1: Tính tổng của các số từ 1 đến 10:

```
Dim i As Integer
Dim Tong As Integer
   Tong = 0
   For i = 1 To 10
         Tong = Tong + i
   Next
Debug.Print ("Tong = " & Tong)
```

- Ví dụ 2: Tính tổng của các số chẵn từ 10 đến 1:

```
Dim i As Integer
Dim Tong As Integer
   Tong = 0
   For i = 10 To 1 Step 2
         Tong = Tong + i
   Next
Debug.Print ("Tong = " & Tong)
```

8.4. Cấu trúc lặp không xác định

Thực hiện một khối lệnh với số lần lặp không định trước và chỉ kết thúc quá trình lặp này khi một biểu thức điều kiện được thỏa mãn *(biểu thức điều kiện có giá trị Boolean: True hoặc False)*.

a. Lặp trong khi điều kiện đúng (True)

- Cú pháp

```
Do While <điều kiện>
   <khối lệnh>
Loop
```

Trong đó:

+ <điều kiện>: là biểu thức logic trả về giá trị True hoặc False.

+ <khối lệnh>: có thể là một lệnh hoặc nhiều lệnh.

- Ý nghĩa

Chương trình sẽ thực hiện kiểm tra <điều kiện>, nếu điều kiện đúng thì thực hiện <khối lệnh>, nếu sai thì kết thúc. Khối lệnh được thực hiện lặp cho đến khi điều kiện sai thì dừng.

b. Lặp trong khi điều kiện sai (False)

- Cú pháp

```
Do
   <khối lệnh>
Loop Until <điều kiện>
```

- Ý nghĩa

Chương trình thực hiện <khối lệnh> trước rồi mới kiểm tra <điều kiện>, nếu điều kiện đúng thì sẽ kết thúc, nếu sai thì thực hiện lặp lại khối lệnh. Khối lệnh sẽ được thực hiện lặp cho đến khi điều kiện đúng thì dừng.

- Chú ý

Đối với cấu trúc lệnh lặp không xác định, trong khối lệnh thực hiện luôn phải có ít nhất một lệnh làm thay đổi giá trị của biến điều kiện nếu không chương trình sẽ bị lặp vô hạn không kết thúc được.

9. Chương trình con

Chương trình con là những đoạn chương trình (module) được viết để giải quyết một công việc hoặc một phần công việc nào đó. Trong Visual Basic, chương trình con có hai dạng là hàm (Function) và thủ tục (Sub).

Mục đích sử dụng chương trình con:

- Chia nhỏ chương trình lớn thành nhiều phần logic tránh viết lặp đi lặp lại, tránh rườm rà.
- Dễ dàng kiểm tra xác định tính đúng đắn của thuật toán, giúp gỡ rối, gỡ lỗi chương trình một cách dễ dàng.
- Có thể được sử dụng lại trong nhiều ứng dụng khác

9.1. Hàm (Function)

Hàm là một chương trình con có kết quả trả về là một giá trị cụ thể và giá trị đó được gán vào tên hàm.

- Cú pháp khai báo

```
Function <tên hàm>([tham số]) As <kiểu dữ liệu>
   <khối lệnh>
   <Tên hàm> = <kết quả trả về>
End Function
```

Trong đó:

+ <tên hàm>: theo quy tắc đặt tên

+ [tham số]: có thể có hay không. (Nếu có nhiều tham số thì mỗi tham số phân cách nhau dấu phẩy. Nếu không xác định kiểu tham số thì tham số có kiểu Variant.)

+ <kiểu dữ liệu>: là kiểu dữ liệu của kết quả do hàm trả về.

- Cú pháp gọi hàm thực thi

<Đối tượng> = <Tên hàm>[(Tham số)]

- Ví dụ: Hàm tính diện tích hình chữ nhật

```
Function Dien_Tich(rong As Double, dai As Double) as Double
    Dien_Tich=rong*dai
End Function
```

9.2. Thủ tục (Sub)

Thủ tục là một chương trình con không trả về một giá trị khi được gọi.

- Cú pháp khai báo

```
Sub <tên thủ tục> ([tham số])
    <khối lệnh>
End Sub
```

Trong đó:

+ <tên thủ tục>: theo quy tắc đặt tên

+ [tham số]: có thể có hoặc không. *(Nếu có nhiều tham số thì mỗi tham số phân cách nhau bởi dấu phẩy)*

- Gọi thủ thục

Để gọi thủ tục sử dụng một trong hai cách sau:

```
<Tên thủ tục>([Tham số])
Call <Tên thủ tục> ([Tham số thực tế])
```

- Ví dụ: Thủ tục tính diện tích hình chữ nhật

```
Sub Dien_Tich(rong as Double, dai as Double, Dt as Double)
   Dt=rong*dai
End Sub
```

10. Các hộp thoại thông dụng

10.1. Hộp thoại thông báo (Message Box)

Hộp thoại thông báo là dạng hộp thoại đơn giản nhất được dùng để hiện cảnh báo hoặc thông báo thông tin cho người dùng và yêu cầu người dùng phải phản hồi bằng một thao tác như bấm chuột.

a. Dạng thủ tục

Hộp thoại chỉ xuất hiện thông báo

MsgBox "Thông báo", [Biểu tượng]+[Nút lệnh],"Tiêu đề"

b. Dạng hàm

Hộp thoại tương tác với người dùng

<Biến> = MsgBox("Thông báo",[Biểu tượng]+[Nút lệnh],"Tiêu đề")

Trong đó:

+ Thông báo: nội dung thông tin chính sẽ hiển thị trên hộp thoại.

+ Biểu tượng: kiểu hình hiển thị trên hộp thoại.

+ Nút lệnh: kiểu các nút nhấn sẽ được hiển thị trên hộp thoại.

+ Tiêu đề: nội dung hiển thị trên thanh tiêu đề của hộp thoại.

+ Biến: lưu giá trị nút nhấn, có kiểu Integer

- Các kiểu biểu tượng

Hằng số	Giá trị	Diễn giải
vbCritical	16	Dùng cho những thông báo lỗi
vbQuestion	32	Dùng cho những câu hỏi yêu cầu chọn
vbExclamation	48	Dùng cho các thông báo của chương trình.
vbInformation	64	Dùng cho các thông báo cung cấp thông tin.

- Các kiểu nút bấm

Hằng số	Giá trị	Diễn giải
vbOKOnly	0	Chỉ hiển thị nút OK
vbOKCancel	1	Hiển thị 2 nút OK và Cancel.
vbAbortRetryIgnore	2	Hiển thị các nút Abort, Retry, và Ignore.
vbYesNoCancel	3	Hiển thị các nút Yes, No, và Cancel.

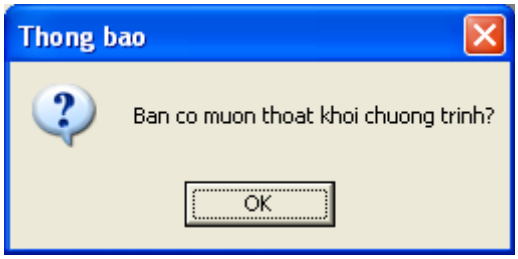
vbYesNo	4	Hiển thị 2 nút Yes và No.
vbRetryCancel	5	Hiển thị 2 nút Retry và Cancel.

- Các hằng nút bấm

Hằng số	Giá trị	Nút bấm
vbOK	1	OK
vbCancel	2	Cancel
vbAbort	3	Abort
vbRetry	4	Retry
vbIgnore	5	Ignore
vbYes	6	Yes
VbNo	7	No

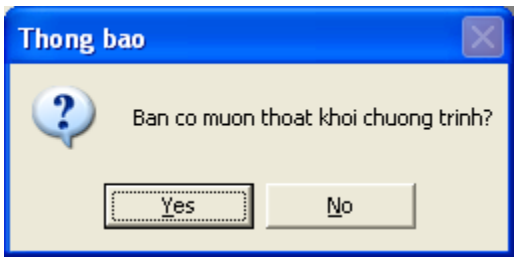
- Ví dụ 1: Loại hộp thoại chỉ xuất hiện thông báo

```
MsgBox "Ban co muon thoat khoi chuong trinh?", vbOKOnly + vbQuestion, "Thong bao"
```



- Ví dụ 2: Loại tương tác với người dùng

```
Dim tl As Integer
    tl = MsgBox("Ban co muon thoat khoi chuong trinh?", vbYesNo + vbQuestion, "Thong bao")
If tl = 6 Then
    End
End If
```



10.2. Hộp thoại nhập liệu (Input Box)

Hộp thoại nhập liệu là dạng hộp thoại cho phép nhận thông tin từ phía người sử dụng. *(Tuy nhiên trong các ứng dụng, hộp nhập rất ít khi được dùng vì: không*

thể kiểm tra thông tin do người dùng nhập vào khi mà chưa ấn Enter và thông tin được nhập là rất ít.)

- Cú pháp

<Biến>=InputBox(“Thông báo”,[Biểu tượng]+[Nút lệnh],“Tiêu đề”)

Trong đó:

+ Thông báo: nội dung thông tin chính sẽ hiển thị trên hộp thoại.

+ Biểu tượng: kiểu hình hiển thị trên hộp thoại.

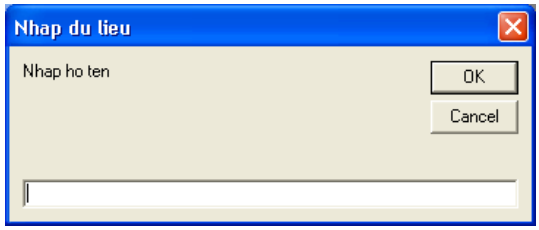
+ Nút lệnh: kiểu các nút nhấn sẽ được hiển thị trên hộp thoại.

+ Tiêu đề: nội dung hiển thị trên thanh tiêu đề của hộp thoại.

+ Biến: lưu giá trị nhập vào hộp thoại và dữ liệu có kiểu String

- Ví dụ

```
Text1.Text = InputBox("Nhap ho ten", "Nhap du lieu")
```



11. Gỡ lỗi và bẫy lỗi trong VBA IDE

Không một chương tình nào là không có lỗi. Tuy nhiên, giảm khả năng lỗi đến mức tối thiểu là có thể làm được

11.1.Một số giải pháp giảm lỗi

- Thiết kế cẩn thận, đặt tên đối tượng theo quy tắc gợi nhớ.
- Ghi ra các vấn đề quan trọng và cách giải quyết cho từng phần.
- Thực hiện đúng các bước lập trình.
- Ghi ra từng thủ tục và mục đích của nó.
- Chú thích rõ ràng trong chương trình
- Chạy thử chương trình từng bước
- Một trong những nguyên nhân gây lỗi là gõ sai tên biến hoặc nhầm lẫn điều khiển. Dùng “Option Explicit” để tránh trường hợp này.

11.2. Đối tượng Err

Là đối tượng do Visual basic cung cấp sẵn để thông báo các lỗi gặp trong chương trình. Đối tượng này có một số thuộc tính cơ bản sau:

Thuộc tính	**Giải thích**
Number	Số hiệu lỗi
Description	Chú thích tóm tắt về lỗi
Source	Tên đối tượng gây ra lỗi

11.3. Một số kỹ thuật gỡ lỗi

a. Dừng chương trình

Có thể tạm dừng chương trình bằng cách chọn Break từ menu Run hoặc nhấn trên thanh công cụ.

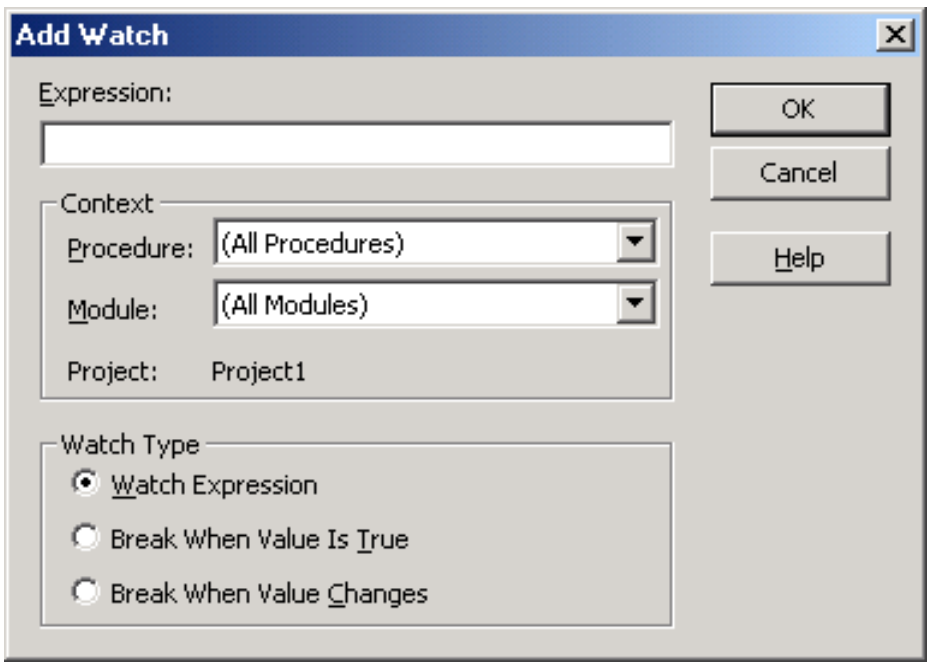
Hoặc nhấn trên tổ hợp phím Ctrl-Break.

Hoặc có thể đặt dòng lệnh Stop trong chương trình.

b. Cửa sổ Immediate

Cửa sổ này cho phép ta xem các giá trị của các biến trong form khi chạy gỡ lỗi. Từ đó phát hiện ra các đối tượng gây lỗi.

c. Cửa sổ Watch



Hiển thị các giá trị của một biến, thuộc tính hay biểu thức bất kỳ. Thậm chí có thể buộc chương trình tạm ngưng sau một số lần lặp.

d. Đi qua từng dòng chương trình

Sử dụng thanh Debug

Thứ tự nút bấm từ trái sang phải như sau:

- *Start*: thi hành chương trình
- *Break*: tạm dừng chương trình
- *End*: Kết thúc chương trình
- *BreakPoint*: Điểm đánh dấu dòng lệnh để tạm dừng chương trình. *(Nút này được sử dụng để bật tắt chế độ breakpoint. Khi có lỗi xảy ra và ta chưa khoanh được khu vực nghi ngờ, thì Breakpoint là giải pháp tốt nhất để cô lập vùng chương trình bị lỗi)*
- *Step Into*: Nếu dòng lệnh hiện hành đang gọi một thủ tục, nhấn F8 sẽ nhảy vào bên trong thủ tục.
- *Step Over*: Nếu dòng lệnh hiện hành đang gọi một thủ tục, nhấn Shipt-F8 sẽ chạy qua thủ tục.
- *Step Out*: Nếu điểm dừng đang ở trong một thủ tục, nhấn Ctrl-Shift-F8 sẽ chạy hết thủ tục và dừng ở dòng kế tiếp sau lệnh gọi thủ tục

11.4. Bẫy lỗi

Lệnh *On Error* dùng trong hàm hay thủ tục báo cho Visual basic biết cách xử lý khi lỗi xảy ra.

On Error GoTo <Nhãn>: sử dụng thuật bẫy lỗi để có thể kiểm soát được lỗi. <Nhãn> là tên thủ tục xử lý lỗi được gọi đến.

On Error Goto 0: tắt xử lý lỗi.

On Error Resume next: bỏ qua lỗi, trả chương trình về dòng lệnh ngay sau dòng lệnh sinh lỗi.

12. Thực hành

13. Kiểm tra

Bài 2: LẬP TRÌNH TRÊN MICROSOFT EXCEL

Lập trình tạo các Macro hỗ trợ cho các công việc xử lý trên bảng tính bằng phần mềm Microsoft Excel.

Thời gian:23 giờ

1. Khái niệm về Macro trong Excel

Khi làm việc trong Excel, có những tình huống mà người sử dụng phải lặp đi lặp lại rất nhiều thao tác để thực hiện các nhiệm vụ tương tự nhau. Thay vì mất thời gian lặp đó, khi thiết kế Excel, Microsoft đã đưa ra khái niệm Macro để có thể gói gọn tất cảcác thao tác ấy vào một thao tác duy nhất.

Macro là tập hợp các lệnh và hàm được lưu trữ trong một mô-đun mã lệnh của VBA nhằm thực hiện một nhiệm vụ nào đó. Macro có thể được tạo bằng cách:

- Excel sẽ tự ghi lại thao tác của người dùng khi làm việc trên nó *(Macro dạng kịch bản)* và khi gọi Macro này, Excel sẽ tự động lặp lại toàn bộ các thao tác trên;
- Người dùng tự viết các đoạn mã lệnh để thực hiện các thao tác tương ứng.

Sau khi được tạo ra, mỗi khi thực thi Macro, tất cả các thao tác đã được lưu trong Macro sẽ được thực hiện tự động.

Về thực chất, Macro là một chương trình con dạng thủ tục (Sub) với từ khoá Public. Tuy nhiên, khác với các thủ tục khác, Macro là thủ tục không có tham số. Chính vì vậy, tất cả các thủ tục với từ khoá Public và không có tham số đều được xem là Macro và sẽ được hiển thị trong trình quản lý Macro của Excel

2. Quản lý Macro

2.1. Tạo Macro

a. Tạo Macro theo kịch bản

Đây là cách tạo Macro dễ dàng nhất, theo cách này, người sử dụng sẽ phải chuẩn bị trước tất cả các thao tác sẽ thực hiện (xây dựng một kich bản), sau đó yêu cầu Excel bắt đầu ghi Macro, người dùng sẽ lần lượt thực hiện

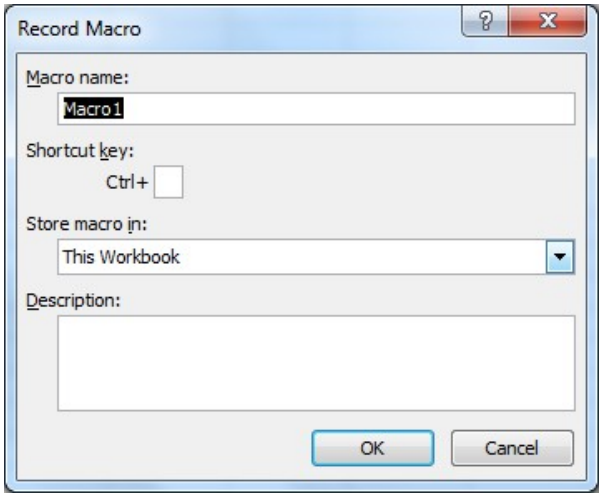
các thao tác theo kịch bản, Excel sẽ ghi nhận các thao tác và tự động chuyển từng thao tác thành các đoạn mã lệnh VBA tương ứng, đoạn mã lệnh này sẽ được lưu lại trong tệp XLS và mặc định là trong Module1.

Chú ý nếu trong quá trình ghi Macro, người sử dụng thực hiện không đúng theo kịch bản dự định (thực hiện sai, bị lỗi) thì Macro cũng sẽ thực hiện lỗi như thế. Vì vậy khi thực hiện thu Macro phải chuẩn bị tốt và cố gắng thực hiện các thao tác một cách thuần thục, chính xác.

Các bước thực hiện:

B1. Khởi động Macro

Trong cửa sổ làm việc của Excel, Mở thực đơn View, chọn Macros → Record Macros (*Excel 2003: Tools → Macro → Record New Macro*) xuất hiện cửa sổ:



B2. Nhập thông tin cho Macro

Macro name: nhập tên Macro

Shortcut key: tạo phím tắt

Store macro in: chọn nơi lưu trữ và phạm vi sử dụng Macro. (This Workbook: tác dụng trên tệp đang làm việc; Personal Macro Workbook: tác dụng trên tất cả các tệp Excel)

Description: mô tả về Macro (có thể bỏ qua)

OK

B3. Ghi Macro

Thực hiện các thao tác mà sau này sẽ được lặp lại khi Macro kịch bản thực thi

B4. Kết thúc Record

Trong cửa sổ làm việc của Excel; Mở thực đơn View, chọn Macros → Stop Record (*Excel 2003: Tools → Macro → Stop Record*)

Hoặc trên thanh công cụ Stop Recording, nhấn chọn biểu tượng Stop Record.

Sau khi kết thúc quá trình tạo Macrro theo kịch bản, Excel sẽ tự động phát sinh đoạn mã lệnh cho thủ tục Macrro.

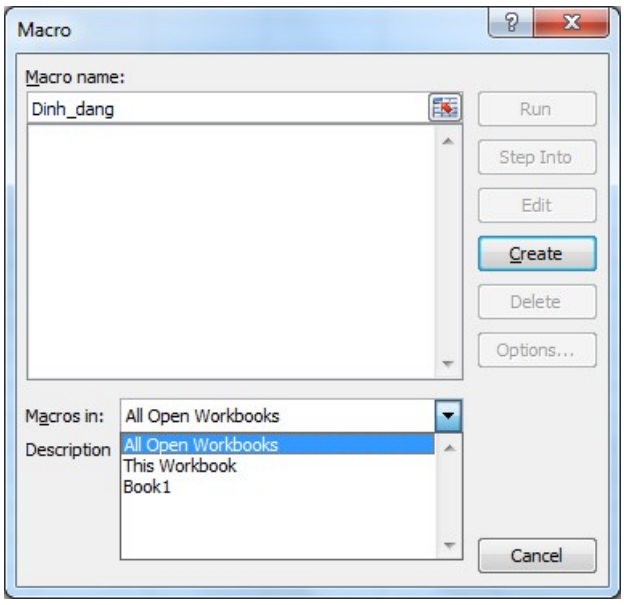
b. Tạo Macro bằng VBA

Trong thực tế, Macro kịch bản không thể đáp ứng được mọi nhu cầu, thông thường nó chỉ đáp ứng tốt những yêu cầu vềthao tác cơ bản khi tương tác với Excel. Để khắc phục nhược điểm này, người dùng có thể viết các đoạn mã lệnh riêng với VBA đểtạo ra các Macro có khả năng đáp ứng được nhu cầu của mình. Như vậy, ngoài cách tạo Macro theo kịch bản, còn có thể tạo Macro bằng cách lập trình trong VBA IDE.

Các bước thực hiện:

B1. Khởi động VBA trong Excel

Trong màn hình chính của Excel, Mở thực đơn View, chọn Macros → View Macros → xuất hiện hộp thoại Macro



+ Macro name: nhập tên Macro (theo quy tắc đặt tên hàm, thủ tục)

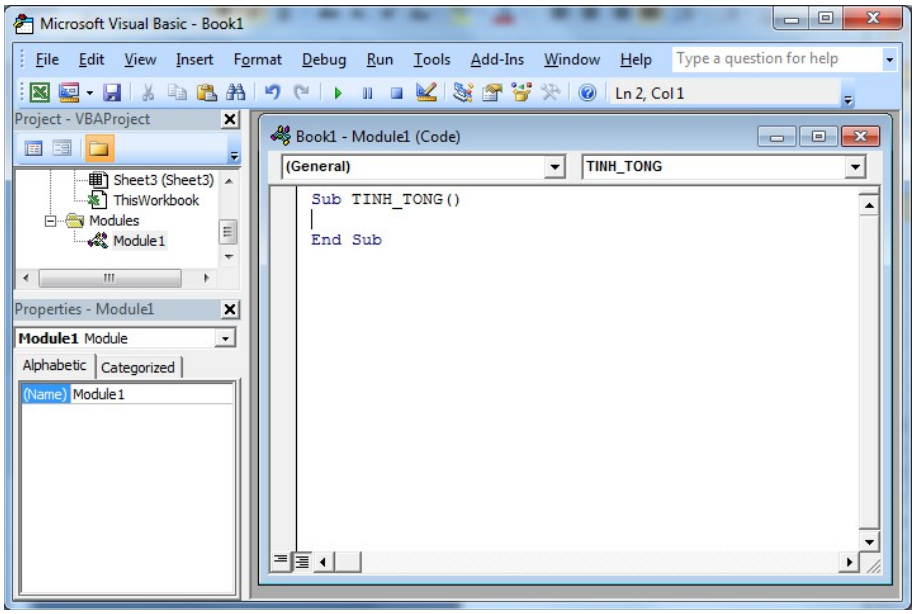
+ Macro in: chọn nơi lưu trữ và phạm vi sử dụng Macro (All Open Workbooks: tất cả các tệp Excel; This Workbook: trên bảng tính hiện tại; Book1: trên tệp Excel hiện tại)

+ Create: thực hiện tạo Macro → xuất hiện cửa sổ soạn thảo mã lệnh

(*Excel 2003: Tools →Macro →Visual Basic Editor (Hoặc Alt + F11)*

Trong màn hình của VBA IDE vừa được hiển thị, chọn Insert → Module. Trong cửa sổ Module, chọn Insert → Procedure → Nhập tên Macro → OK → xuất hiện cửa sổ soạn thảo mã lệnh

B2. Soạn thảo mã lệnh



B3. Đóng Macro

Sau khi nhập xong đoạn mã lệnh, chọn File → Close and Return to Microsoft Excel để trở về màn hình chính của Excel

CHÚ Ý: Mỗi Macro đều có một tên riêng và tên này là duy nhất trong một tệp Excel (Workbook).

2.2. Lưu tệp chứa Macro

File → Save → xuất hiện hộp thoại

+ Save in: chọn vị trí lưu tệp

+ File name: đặt tên tệp

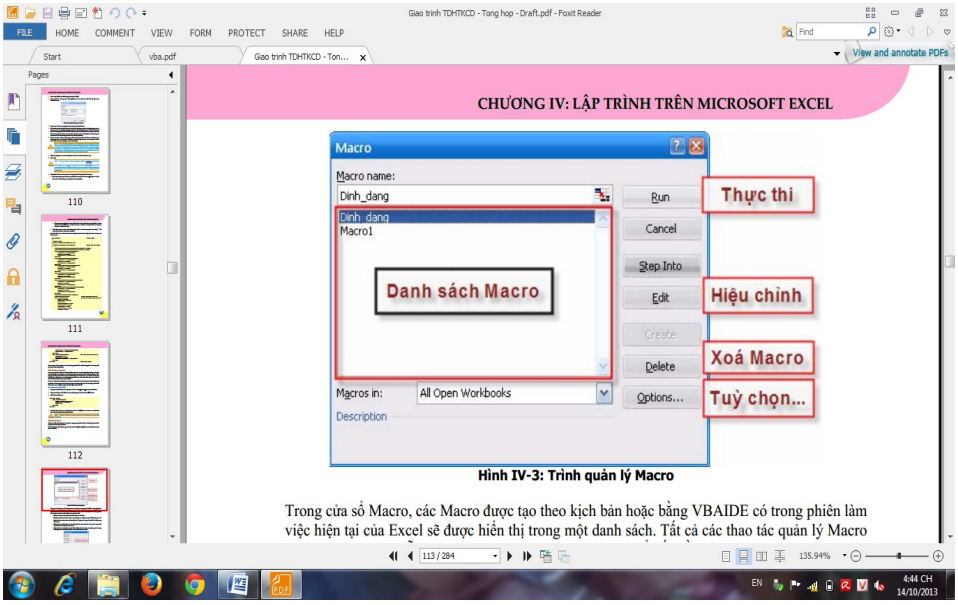
+ Save as type: Excel Macro-Enabled Workbook (*.xlsm)

Chú ý: Để lưu Workbook có chứa các hàm viết bằng VBA, phải dùng loại file: Excel Macro-Enabled Workbook (.xlsm))*

2.3. Quản lý Macro

**** Hiển thị trình quản lý Macro:***

Trong màn hình chính của Excel, Mở thực đơn View, chọn Macros → View Macros → xuất hiện hộp thoại Macro



**** Thực thi Macro***

Hiển thị trình quản lý Macro → chọn tên Macro muốn thực thi → thực hiện lệnh Run

**** Hiệu chỉnh Macro***

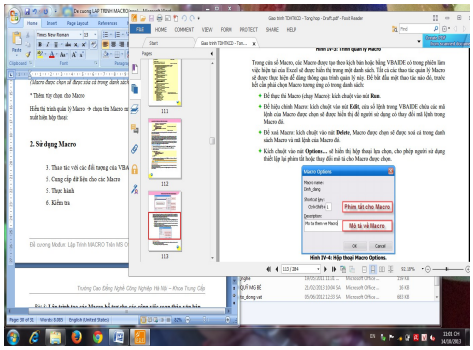
Hiển thị trình quản lý Macro → chọn tên Macro muốn hiệu chỉnh → thực hiện lệnh Edit → xuất hiện cửa sổ lệnh VBA IDE → thay đổi mã lệnh

**** Xóa Macro***

Hiển thị trình quản lý Macro → chọn tên Macro muốn xóa → thực hiện lệnh Delete *(Macro được chọn sẽ được xóa cả trong danh sách Macro và mã lệnh của macro đó)*

**** Thêm tùy chọn cho Macro***

Hiển thị trình quản lý Macro → chọn tên Macro muốn xóa → thực hiện lệnh Options → xuất hiện hộp thoại:



+ Shortcut key: thay đổi phím tắt

+ Description: thêm mô tả về Macro

3. Sử dụng Macro

Việc sử dụng các Macro đã được tạo, thực chất là thực thi đoạn mã lệnh tạo nên Macro đó

3.1. Thực thi macro bằng phím tắt

Trong quá trình tạo hoặc quản lý Macro, người sử dụng có thể gán một phím tắt cho Macro đó. Và để thực thi Macro, người dùng chỉ cần nhấn tổ hợp phím tắt đã gán cho Macro.

Ví dụ: Ở phần tạo Macro định dạng, người dùng đã gán cho Macro này một tổ hợp phím tắt là CTRL+SHIFT+L, do vậy, để thực thi Macro, người sử dụng chỉ cần chọn vùng dữ liệu để định dạng bảng, sau đó nhấn tổ hợp phím CTRL+SHIFT+L.

3.2. Thực thi Macro thông qua trình quản lý

Trong màn hình chính của Excel, Mở thực đơn View, chọn Macros → View Macros → xuất hiện hộp thoại Macro → chọn tên Macro muốn thực thi → thực hiện lệnh Run

3.3. Thực thi macro trực tiếp từ VBA IDE

Cách thực thi Macro trực tiếp từVBAIDE rất thích hợp khi người sửdụng muốn thử nghiệm ngay Macro trong quá trình xây dựng nó.

Để thực thi Macro nào đó trong VBAIDE, cần thực hiện như sau:

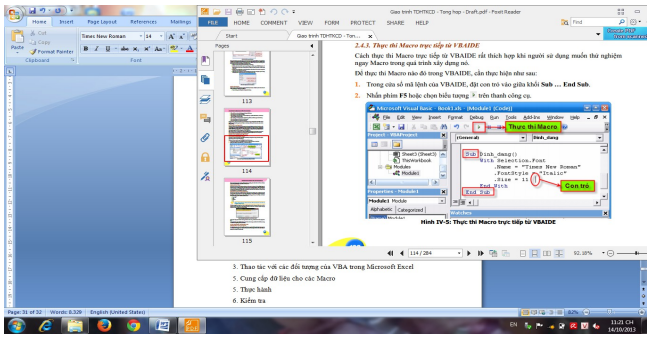
Trong cửa sổmã lệnh của VBAIDE, đặt con trỏvào giữa khối

Sub

…

End Sub.

Nhấn phím F5 hoặc chọn biểu tượng Run trên thanh công cụ.

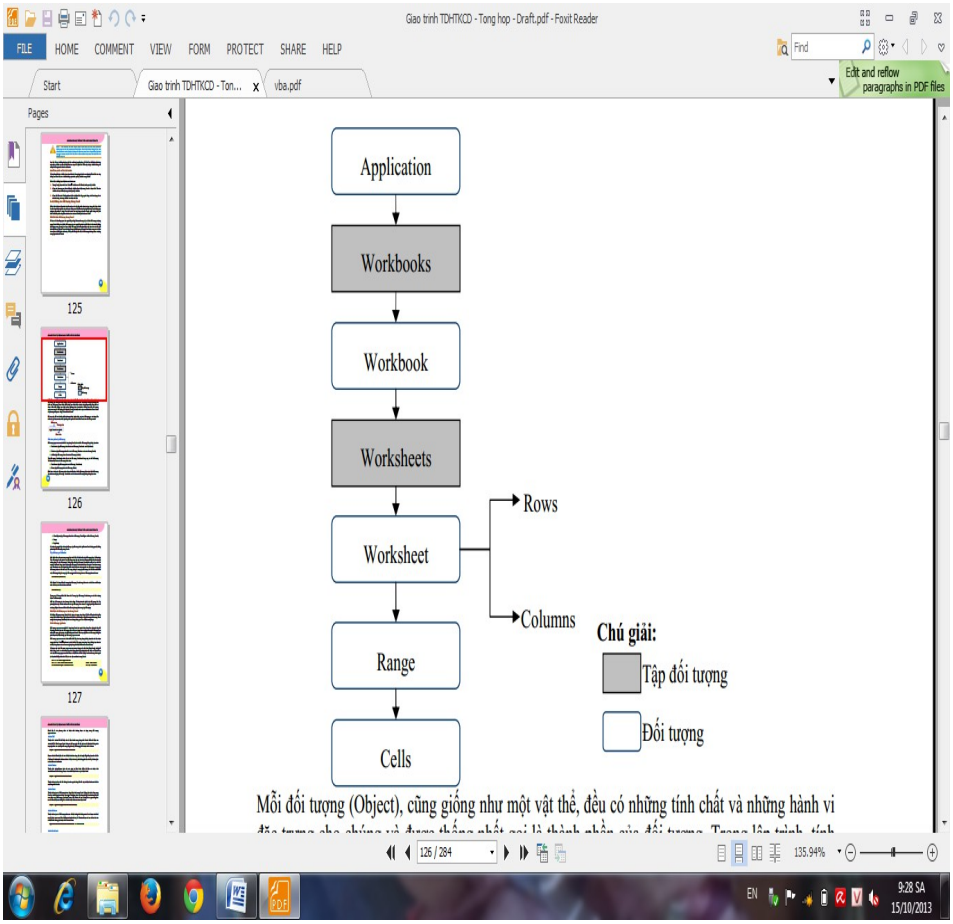


4. Thao tác với các đối tượng của VBA trong Microsoft Excel

4.1. Mô hình đối tượng trong MS Excel

Để tạo cái nhìn tổng quan cho người lập trình, Microsoft cung cấp mô hình đối tượng sử dụng trong Excel. Nhờ có mô hình đối tượng này mà người lập trình có thể hiểu rõ cấu trúc hệ thống đối tượng trong Excel, tìm được đúng đối tượng khi cần thực hiện một thao tác nào đó.

a. Cấu trúc phân cấp đối tượng



Mỗi đối tượng (Object), cũng giống như một vật thể, đều có những tính chất và những hành vi đặc trưng cho chúng và được thống nhất gọi là thành phần của đối tượng. Trong lập trình, tính chất của đối tượng được biểu diễn thông qua khái niệm thuộc tính (properties), còn hành vi được biểu diễn thông qua khái niệm phương thức (methods).

Để truy cập đến các thành phần (phương thức, thuộc tính, …) của đối tượng, ta sử dụng cú pháp:

<Đối tượng>. <Thành phần>

Ví dụ: Application.Quit

b. Tập đối tượng (Collection)

Tập đối tượng là một nhóm các đối tượng cùng lớp với nhau (và đương nhiên, bản thân tập đối tượng cũng là một đối tượng). Ví dụ như tập đối tượng Workbooks chứa tất cả các đối tượng Workbook đang được mở hay tập đối tượng Worksheets chứa tất cả các Worksheet trong một Workbook

nào đó. Người lập trình có thể thao tác trên toàn bộ các đối tượng có trong tập đối tượng hoặc có thể trên một đối tượng riêng lẻ trong tập đối tượng đó.

Để tham chiếu đến một đối tượng riêng lẻ trong tập đối tượng, có thể sử dụng tên của đối tượng theo cách sau:

<Tập đối tượng>.<Tên đối tượng>

Ví dụ: Worksheets("Sheer1")

Hoặc: Worksheets(1)

4.2. Đối tượng Application

a. Giới thiệu

Đối tượng Application chính là ứng dụng Excel mà người dùng đang làm việc trên đó, mỗi lần chạy Excel sẽ có một đối tượng Application được tạo ra. Application là đối tượng cao nhất (đối tượng gốc) trong cây đối tượng của Excel. Việc truy cập đến các đối tượng khác, cần phải được thực hiện thông qua đối tượng Application.

Việc tạo mới một đối tượng Application tương đương với việc khởi động Excel, do đó, để khởi động Excel từ môi trường lập trình khác, người lập trình phải viết đoạn mã lệnh để tạo mới một đối tượng Application.

Ví dụ: *Đoạn mã lệnh sau sẽ khởi động Excel từ chương trình ngoài (lập trình trên VB) và mở một workbook trong Excel:*

```
Dim xl As Excel.Application
Set xl = New Excel.Application          'Khởi động Excel
xl.Workbooks.Open "newbook.xls"   'Mở một Workbook
```

b. Một số phương thức và thuộc tính

ActiveCell: thể hiện cho ô hiện thành trong bảng tính Excel. Kiểu dữ liệu của ActiveCell là kiểu Range

Ví dụ :

MsgBox Application.ActiveCell.Address ' trả về địa chỉ ô ActiveSheet

Thuộc tính này trả về đối tượng sheet đang hiện hành trong Excel. Cũng cần chú ý rằng trong Excel có nhiều loại sheet khác nhau như Worksheet (loại hay dùng nhất), Chartsheet…,

Đoạn mã sau sử dụng hàm TypeName, hàm trảvề kiểu dữliệu của biến, để từ đó biết được kiểu của sheet hiện hành:

MsgBox TypeName(Application.ActiveSheet)

ActiveWindow: Thuộc tính này trảvề đối tượng chứa cửa sổhiện hành, nếu không cửa sổnào được mở thì sẽ trảvềgiá trị Nothing. Kiểu dữ liệu của thuộc tính này là Window.

Ví dụ: Đoạn mã sau sẽthu nhỏ cửa sổ hiện hành thông qua thuộc tính WindowState:

Application.ActiveWindow.WindowState = xlMinimized

ActiveWorkbook: Thuộc tính này trả về đối tượng chứa workbook nằm trong cửa sổ hiện hành (tệp XLS đang làm việc), nếu không có cửa sổ nào được mở hoặc cửa sổ đó là cửa sổ không chứa workbook (như cửa sổ Info, Clipboard,...) thì sẽ trảvề giá trị Nothing. Kiểu dữ liệu của thuộc tính này là Workbook.

Ví dụ: Đoạn mã lệnh sau sẽ hiển thị tên của workbook hiện hành:

MsgBox Application.ActiveWorkbook.Name

Thuộc tính ActiveWorkbook và ActiveWindow rất dễ nhầm lẫn với nhau. Thoạt nhìn, mỗi workbook cũng giống như một cửa sổ trong Excel, nhưng thực chất không phải vậy. Để rõ hơn sự khác biệt giữa workbook và cửa sổ, ta tạo thêm một cửa sổmới bằng cách chọn trình đơn Window →New Window. Cửa sổ mới được tạo có nội dung giống như cửa sổban đầu, nhưng người dùng có thể lựa chọn những vùng khác nhau trên hai cửa sổ.

4.3. Làm việc với Workbook

4.4. Làm việc với đối tượng Window

4.5. Làm việc với đối tượng Range và Cells

5. Cung cấp dữ liệu cho các Macro

Bài 3: Lập trình tạo các Macro hỗ trợ cho các công việc soạn thảo văn bản bằng phần mềm Microsoft word

Thời gian:23 giờ

1. Khái niệm về Macro trong Microsoft Word

2. Quản lý Macro

3. Sử dụng Macro

4. Thao tác với các đối tượng của VBA trong Microsoft Word

5. Cung cấp dữ liệu cho các Macro