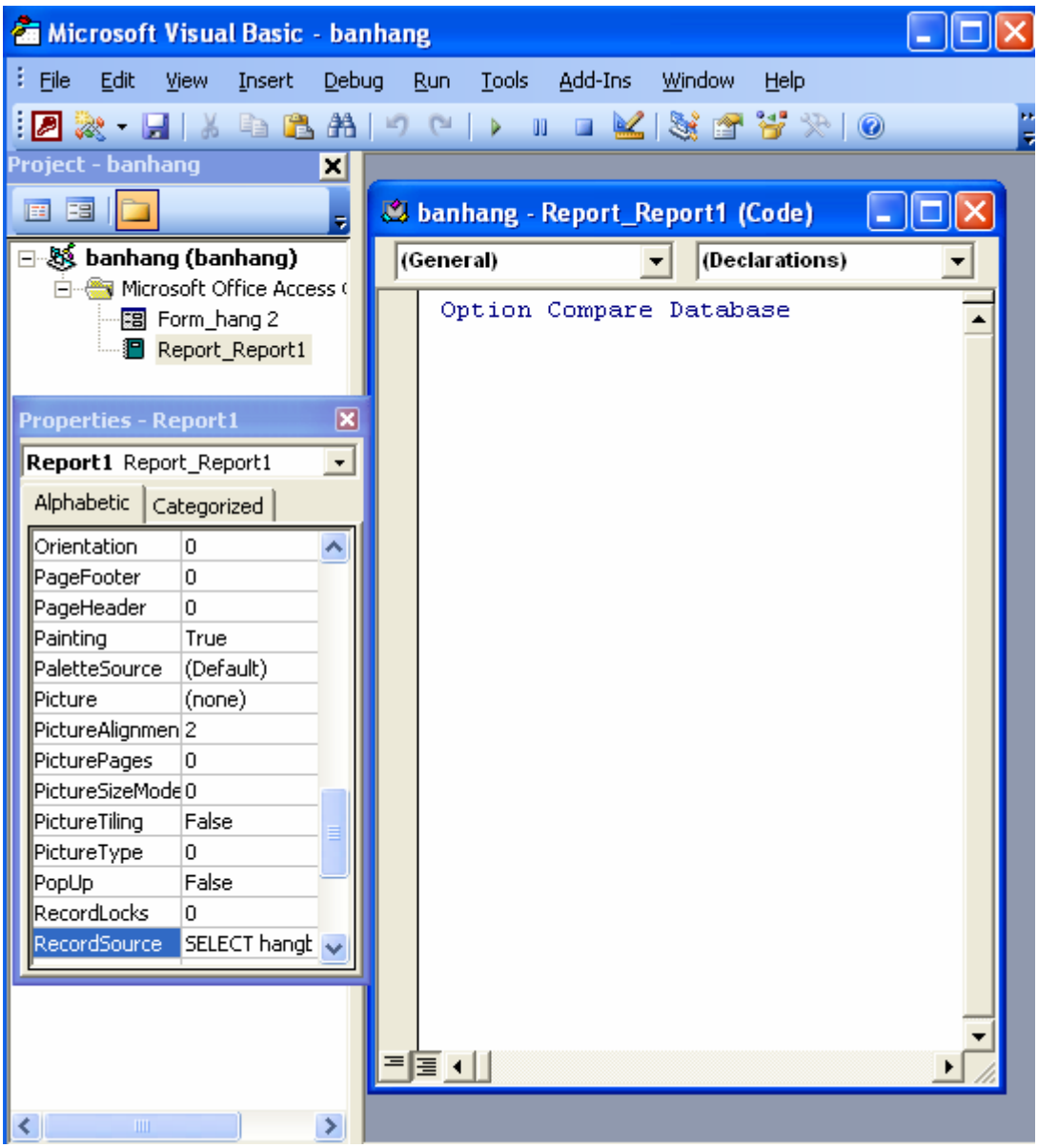
CHƯƠNG 5
LẬP TRÌNH VBA CĂN BẢN

1. Môi trường lập trình VBA

VBA – Visual Basic for Application để giúp người dùng có thể tạo ra các tuỳ biến mạnh hơn, thân thiện hơn với trong công việc của mình. Hơn thế nữa VBA trên Access đã thể hiện tính chuyên nghiệp trong lập trình, phần nào biến được một CSDL đơn giản trở thành những sản phẩm đóng gói thương mại.

Màn hình làm việc ngôn ngữ VBA thường có dạng:



** Hệ thống thực đơn và thanh công cụ*

Cũng như bất kỳ môi trường làm việc nào đều có hệ thống thực đơn và thanh công cụ đi kèm. Trên đó có chứa các lệnh để gọi, thi hành hoặc thiết lập các điều khiển cần thiết.

** Cửa sổ Project Explorer*

Có rất nhiều các thành phần có thể lập trình được bởi VBA như: Forms, Reports, Modules. Cửa sổ Project Explorer là cây phân cấp lớp các đối tượng có chứa mã lệnh VBA, đồng thời giúp lập trình viên dễ dàng trong việc viết (coding) cũng như quản lý các mã lệnh VBA đã viết.

** Cửa sổ viết lệnh*

Cửa sổ viết lệnh là nơi soạn thảo các dòng lệnh VBA. Mỗi cửa sổ sẽ chứa toàn bộ mã lệnh cho một đối tượng như: Forms, Reports, Modules. Trong mỗi cửa sổ có thể có nhiều phần được viết lệnh, mỗi phần có thể là nội dung một khai báo, một chương trình con, nội dung một thủ tục đáp ứng sự kiện.

** Cửa sổ Intermediate*

Cửa sổ Intermediate là nơi giúp thi hành trực tiếp một câu lệnh nào đó, rất hữu dụng trong việc gỡ lỗi phần mềm.

2. Kiểu dữ liệu - Biến - Hằng

2.1 Kiểu dữ liệu cơ bản

a, Boolean

Kiểu lô gíc, tương tự kiểu Boolean trên Pascal. Kiểu này chiếm 2 byte bộ nhớ; chỉ nhận một trong 2 giá trị là: Yes – No hoặc True – False hoặc đôi khi thể hiện dưới dạng số 0 tương đương với False, True tương ứng với bất kỳ số nào khác 0.

Khi lập trình CSDL, kiểu Boolean tương ứng với kiểu Yes/No trong bảng dữ liệu.

b, Byte

Kiểu số nguyên dương trong phạm vi từ 0..255. Kiểu này chiếm 1 byte bộ nhớ.

c, Integer

Kiểu nguyên, có giá trị trong khoảng -32768...32767. Kiểu này chiếm 2 bytes bộ nhớ.

d, Long

Kiểu số nguyên *dài*, có giá trị trong khoảng 2,147,483,648 .. 2,147,483,647.

Kiểu này chiếm 4 bytes bộ nhớ.

e, Single

Kiểu số thực, có giá trị trong khoảng 1.401298E-45 to 3.402823E38. Chiếm 4 bytes bộ nhớ.

f, Double

Kiểu số thực có đợ lớn hơn kiểu Single, có giá trị trong khoảng 4.94065645841247E-324 đến 1.79769313486232E308.

Chiếm 8 bytes bộ nhớ.

g, Currency

Kiểu tiền tệ. Bản chất là kiểu số, độ lớn 8 bytes, có giá trị trong khoảng -922,337,203,685,477.5808 đến 922,337,203,685,477.5807.

Đặc biệt, kiểu này luôn có ký hiệu tiền tệ đi kèm.

h, String

Kiểu xâu ký tự. Kiểu này tương ứng với kiểu String trong Pascal, tương ứng với kiểu Text trong các trường CSDL Access. Độ lớn tối đa 255 bytes tương đương với khả năng xử lý xâu dài 255 ký tự.

i, Variant

Variant là kiểu dữ liệu không tường minh. Biến kiểu này có thể nhận bất kỳ một giá trị nào có thể.

Người ta thường khai báo biến kiểu Variant trong những trường hợp phải xử lý biến đó mềm dẻo. Khi thì biến nhận giá trị kiểu này, khi thì nhận giá trị và xử lý theo kiểu dữ liệu khác.

j, Object

Object là một loại biến kiểu Variant, chiếm dung lượng nhớ 4 bytes, dùng để tham chiếu tới một loại đối tượng (Object) nào đó trong khi lập trình. Tất nhiên muốn khai báo biến Object kiểu nào, phải chắc chắn đối tượng đó đã được đăng ký vào thư viện tham chiếu VBA bởi tính năng **Tool | Reference**. Chúng ta sẽ còn trở lại vấn đề này khi lập trình CSDL.

2.2 Biến và cách sử dụng biến

a. Khai báo biến

Biến (Variable) là thành phần của một ngôn ngữ lập trình, giúp xử lý dữ liệu một cách linh hoạt và mềm dẻo.

Thông thường trong các ngôn ngữ lập trình, mỗi biến khi tồn tại phải được định kiểu, tức là phải nhận một kiểu dữ liệu xác định. Tuy nhiên trong VBA thì không, mỗi biến có thể định kiểu (được khai báo trước khi sử dụng) hoặc không định kiểu (không khai báo vẫn sử dụng được). Trong trường hợp này biến đó sẽ tự nhận kiểu giá trị Variant.

Biến có thể được khai báo bất kỳ ở đâu trong phần viết lệnh của VBA. Tất nhiên, biến có hiệu lực như khai báo chỉ bắt đầu từ sau lời khai báo và đảm bảo phạm vi hoạt động như đã qui định.

Cú pháp khai báo biến:

Dim <tên_biến> **As** <tên_kiểu>

b. Phạm vi biến

Như chúng ta đã biết, mỗi biến sau khi được khai báo nó sẽ nhận một kiểu dữ liệu và có một phạm vi hoạt động, tức là lời khai báo biến chỉ có tác dụng trong những vùng đã được chỉ định; ngoài vùng chỉ định đó biến sẽ không có tác dụng, nếu có tác dụng sẽ theo nghĩa khác (biến cục bộ kiểu Variant chẳng hạn).

*** Biến cục bộ:**

Biến cục bộ được khai báo sau từ khoá **Dim**, nó chỉ có tác dụng trong một chương trình con, cục bộ trong một form hoặc một module nào đó. Dưới đây sẽ chỉ ra 3 trường hợp biến cục bộ này:

- Trong một chương trình con, nếu nó được khai báo trong chương trình con đó;
- Trong cả một Form, nếu nó được khai báo trong phần Decralations của Form đó;
- Trong cả một Reports, nếu nó được khai báo trong phần Decralations của Report đó;
- Trong cả một Modules, nếu nó được khai báo trong phần Decralations của Modules đó;

*** Biến toàn cục:**

Biến toàn cục được khai báo sau cụm từ khoá **Public**, nó có tác dụng trong toàn bộ chương trình (ở bất kỳ chỗ nào có thể viết lệnh). Loại biến này luôn phải được khái báo tại vùng *Decralations* của một Module nào đó.

Trên một tệp Access, không được phép khai báo trùng tên biến toàn cục. Tuy nhiên tên biến cục bộ vẫn có thể trùng tên biến toàn cục, trong trường hợp đó VBA sẽ ưu tiên sử dụng biến cục bộ trong phạm vi của nó.

2.3 Hằng và cách sử dụng hằng

a. Khai báo hằng

Hằng (Constan) là đại lượng có giá trị xác định và không bị thay đổi trong bất kỳ hoàn cảnh nào. Tương ứng với từng kiểu dữ liệu, sẽ có những hằng tương ứng.

Cú pháp khai báo hằng:

Const <tên_hằng> = <giá_trị>

Sau đây là các ví dụ về khai báo các loại hằng:

```
Const ngay = #24/12/2004#
Const phongban = "Tài vụ"
```

```
Const ok = True
```

b. Phạm vi hằng

Tương tự như biến, hằng cũng có những phạm vi hoạt động của nó. Hằng được khai báo trong thủ tục nào, hoặc cục bộ trong form, report hoặc module nào sẽ chỉ có tác dụng trong phạm vi đó.

Muốn hằng có phạm vi toàn cục, phải được khai báo sau từ khoá **Public Const**, tại vùng *Decralations* của một module nào đó như sau:

```
Public Const <tên_hằng> = <giá_trị>
```

3. Các cấu trúc lệnh VBA

Các cấu trúc lệnh là thành phần cơ bản của mỗi ngôn ngữ lập trình. Thông thường các ngôn ngữ lập trình đều có các cấu trúc lệnh như nhau: lệnh xử lý điều kiện, lệnh lặp biết trước số vòng lặp, lệnh lặp không biết trước số vòng lặp,.. Tuy nhiên cách thể hiện (cú pháp) mỗi cấu trúc lệnh có thể khác nhau tuỳ thuộc vào mỗi ngôn ngữ lập trình. Hơn nữa, mỗi ngôn ngữ cũng có thể có một số điểm khác biệt, đặc trưng trong mỗi cấu trúc lệnh.

Cũng giống như nhiều ngôn ngữ lập trình hiện đại khác, các cấu trúc lệnh trong VBA đều tuân thủ các nguyên tắc:

- Có cấu trúc: mỗi cấu trúc lệnh đều có từ khoá bắt đầu và một từ khóa báo hiệu kết thúc.
- Thực hiện tuần tự (loại trừ trường hợp đặc biệt thủ tục Goto <Label>)
- Có khả năng lồng nhau.

3.1 Cấu trúc rẽ nhánh

Cấu trúc rẽ nhánh hay còn gọi là *lệnh lựa chọn.* Tức là nếu một điều kiện nào đó xảy ra sẽ là gì, hoặc trái lại có thể làm gì.

Cú pháp:

```
If <điều kiện> Then
     <thủ tục 1>
[ Else
     <thủ tục 2> ]
End If
```

Ý nghĩa lệnh trên là: **nếu** <*điều kiện*> = *True* **thì** thực hiện các lệnh trong <*thủ*

tục1>. Trái lại thực hiện các lệnh trong <*thủ tục 2*>.

Phần trong cặp dấu ngoặc vuông [..] có thể có hoặc không có trong câu lệnh, tuỳ

thuộc vào mục đích xử lý.

3.2 Cấu trúc lựa chọn

Đây là một loại của cấu trúc lựa chọn. Thông thường hoàn toàn có thể sử dụng If .. End If để thực hiện các xử lý liên quan đến kiểu cấu trúc này, nhưng trong những trường hợp đặc biệt, cấu trúc **Select Case .. End Select** thể hiện được sự tiện dụng vượt trội.

Cú pháp

```
Select Case <biểu thức>
    Case <giá trị 1>
        <thủ tục 1>
    Case <giá trị 2>
        <thủ tục 2>
    ……
    Case <giá trị n>
        <thủ tục n>
    [Case Else
        <thủ tục n+1>]
End Select
```

Trong đó: <Biểu thức> luôn trả về giá trị kiểu vô hướng đếm được như: số nguyên, xâu ký tự, kiểu lô gíc,..

Với cấu trúc này, VBA hoạt động như sau:

(1) Tính giá trị của biểu thức

(2) Kiểm tra <*biểu thức*> = <*giá trị 1*> ?

- Nếu đúng thực hiện <thủ tục 1> và kết thúc lệnh, thực hiện lệnh tiếp theo sau từ khoá End Select.

- Nếu sai, thực hiện tiếp việc so sánh <biểu thức> = <giá trị i> tiếp theo và xử lý tương tự qui trình nêu trên.

(3) Trong trường hợp <*biểu thức*> <> <*giá trị i*>, i=1..n khi đó có 2 khả năng:

- Nếu có tuỳ chọn *Case Else* thì VBA sẽ thực hiện <*thủ tục n+1*>;

- Nếu không có tuỳ chọn *Case Else*, VBA sẽ không thực hiện bất kỳ thủ tục nào đã liệt kê trong vùng Select .. End Select cả mà chuyển tới thực hiện lệnh tiếp theo sau từ khoá End Select.

3.3 Cấu trúc lặp

a, Cấu trúc FOR ... NEXT

For… Next là một cấu trúc lặp biết trước số lần lặp trong VBA, tuy nhiên trong những tình huống đặc biệt, vẫn có thể sử dụng cấu trúc này như cấu trúc không biết trước được số lần lặp.

Cú pháp:

For <*biến chạy*> = <*giá trị 1*> **To** <giá trị 2> [**Step** <*n*>]
    <*thủ tục*>
    [**Exit For**]
Next

Trong đó:

- <*biến chạy*> là biến kiểu vô hướng đếm được, hay dùng nhất là biến kiểu nguyên;

- <*giá trị 1*>, <*giá trị 2*> là các giá trị mà biến chạy sẽ nhận và thực hiện dịch chuyển sau mỗi lần lặp. Có thể dịch chuyển đi 1 đơn vị, có thể dịch chuyển đi nhiều đơn vị một lần, có thể dịch chuyển tiến, cũng có thể dịch chuyển lùi- tất cả điều này tuỳ thuộc vào việc có hay không có tuỳ chọn [Step <n>];

- Nếu có tuỳ chọn *[***Step** <*n*>*]* biến chạy sẽ dịch n đơn vị sau mỗi lần lặp. Khi đó, nếu n>0 dẽ dịch tiến, ngược lại sẽ dịch lùi;

- Mỗi lần lặp, VBA sẽ thực hiện <*thủ tục*> một lần;

- Trong trường hợp đặc biệt nếu gặp phải lệnh **Exit For** trong vòng lặp, ngay lập tức thoát khỏi lệnh lặp và thực hiện lệnh tiếp ngay sau từ khoá Next. Chính **Exit For** đã làm mất đi tính lặp biết trước được số lần lặp.

b, Cấu trúc WHILE ... WEND

While … Wend là một cấu trúc lặp không biết trước số lần lặp trong VBA.

Cú pháp:

```
While <điều kiện>
    <thủ tục>
Wend
```

Trong đó:

- While, Wend là các từ khoá của lệnh lặp;

- Nếu <*điều kiện*> = *True*, các lệnh trong <*thủ tục*> sẽ được thực hiện. Thực hiện xong lại quay lên dòng lệnh While để kiểm tra tiếp <điều kiện>;

- Nếu <*điều kiện*> = *False*, sẽ thoát khỏi vòng lặp và thực hiện lệnh tiếp theo từ khoá **Wend**.

Chú ý: Luôn phải chứng minh được rằng, sau một số hữu hạn lần thực hiện <*thủ tục*>, giá trị của <*biểu thức*> phải là *False* để thoát khỏi vòng lặp. Trong trường hợp không thể thoát khỏi vòng lặp, có nghĩa người lập trình đã mắc phải *lỗi lặp vô hạn*. Có thể dẫn đến chương trình bị treo.

4. Lệnh DoCmd

Chúng ta có thể dùng lệnh **DoCmd** để thi hành các công việc thông thường trên Access thông qua môi trường VBA. Như: dùng DoCmd để có thể mở form, mở report, query, lọc dữ liệu, thi hành macro xử lý bản ghi, ứng dụng,.. Hầu hết các thao tác xử lý trên các đối tượng của Access đều có thể dùng lệnh doCmd để gọi ra thực hiện trong môi trường VBA.

Dưới đây liệt kê một số các phép xử lý của lệnh DoCmd thông dụng:

4.1 Lệnh đóng một đối tượng

Lệnh này để đóng (Close) hoặc giải phóng đối tượng nào đó ra khỏi bộ nhớ. Hay dùng lệnh này để đóng form đang hoạt động hoặc đóng một report đang preview.

Cú pháp như sau:

DoCmd.Close [ObjectType], [ObjectName], [SaveOption]

Trong đó:

ObjectType: chỉ kiểu đối tượng cần đóng. Cụ thể như sau:

- *acForrm:* Đóng form
- *acReport:* Đóng report
- *acQuery:* Đóng query
- *acTable:* Đóng bảng

ObjectName: chỉ tên đối tượng cần đóng;

S**aveOption:** chỉ định tuỳ chọn ghi lại cấu trúc (nếu có sự thay đổi). Cụ thể:

- *SaveNo* Không khi lại
- *SaveYes* Luôn ghi lại
- *SavePromt* Hiển thị hộp thoại nhắc để ghi nếu có sự thay đổi

4.2 Lệnh mở form

Là một lệnh hoàn chỉnh để mở và thiết lập môi trường làm việc cho một form.

Cú pháp như sau:

DoCmd.OpenForm [objectName], [ViewMode], [FilterName], [WhereCondition], [DataMode], [WindowsMode]

Trong đó:

ObjectName: tên form muốn mở

ViewMode: chế độ mở. Cụ thể

- *acDesign* Mở form ra chế độ thiết kế
- *acNormal* Mở form ra để thi hành

FilterName: Đặt lọc

WhereCondition: Giới hạn các bản ghi trong nguồn dữ liệu

DataMode: thiết lập chế độ dữ liệu trên form, cụ thể:

WindowsMode: thiết lập kiểu cửa sổ form là:

- *acDialog:* Kiểu hộp thoại
- *acWindowsNormal:* Kiểu cửa sổ bình thường

4.3 Lệnh mở report

Là một lệnh hoàn chỉnh để mở và thiết lập môi trường làm việc cho một report.

Cú pháp như sau:

DoCmd.OpenReport [objectName], [ViewMode], [FilterName], [WhereCondition], [DataMode], [WindowsMode]

Trong đó:

ObjectName: tên Report muốn mở;

ViewMode: chế độ mở. Cụ thể:

- *acDesign:* Mở Report ra chế độ thiết kế
- *acNormal:* Mở Report ra để thi hành

FilterName: Đặt lọc

WhereCondition: Giới hạn các bản ghi trong nguồn dữ liệu

DataMode: thiết lập chế độ dữ liệu trên Report , cụ thể:

WindowsMode: thiết lập kiểu cửa sổ Report là:

- *acDialog:* Kiểu hộp thoại
- *acWindowsNormal:* Kiểu cửa sổ bình thường

4.4 Lệnh thi hành câu lệnh SQL

Dùng để thi hành một lệnh SQL. Cú pháp như sau:

DoCmd.RunSQL

5. Chương trình con

Chương trình con là một đơn vị mã lệnh VBA, nó có thể chứa tập hợp các câu lệnh nhằm thao tác, tính toán hoặc điều khiển mục đích hoặc dữ liệu nào đó. Trong VBA có 2 loại chương trình con:

- Chương trình con dạng thủ tục, được khai báo bởi từ khoá Sub.
- Chương trình con dạng hàm, được khai báo bởi từ khoá Function.

Về bản chất, 2 loại chương trình con trên đều như nhau: khai báo, tham số và truyền tham số. Tuy nhiên, điểm khác nhau cơ bản là:

- Function luôn trả về một giá trị kiểu vô hướng chuẩn, ví dụ: hàm Date() trả về giá trị ngày hiện tại kiểu Date. Trong Access đã sẵn có rất nhiều các hàm tính toán, chúng được gọi là các build-in fuction. Hơn nữa, người dùng hoàn toàn có thể tạo ra các hàm để sử dụng cho các mục đích riêng loại hàm này gọi là user-define function;
- Sub chỉ thực hiện một số các công việc. Tất nhiên những công việc này hoàn toàn có thể làm thay đổi dữ liệu theo mong muốn trong chương trình. Cũng như Function, Access và VBA sẵn có một thư viện các thủ tục; hơn nữa người dùng cũng có thể tự tạo thêm những thủ tục mới phục vụ việc xử lý dữ liệu theo mục đích riêng. Đặc biệt, Access còn định nghĩa thủ tục đáp ứng sự kiện. Thủ tục này sẽ được tự động gọi ra khi sự kiện đáp ứng bị ảnh hưởng. Chúng ta sẽ trở lại nội dung này qua các ví dụ lập trình VBA.

Tuỳ từng tính huống cụ thể sẽ lựa chọn sử dụng Function hoặc Sub.

5.1 Chương trình con dạng hàm

Cú pháp

Function <*tên hàm*>([<*danh sách các tham số*>]) **As** <kiểu DL hàm>

<*thủ tục*>

End Function

Trong đó:

- **Function, End Function** là các từ khoá bắt buộc khai báo cấu trúc một chương trình con dạng hàm;
- <*tên hàm*> là tên gọi hàm định khai báo. Tên không được chứa dấu cách (space) và các ký tự đặc biệt;

- <*danh sách các tham số*> danh sách các tham số cần thiết cho hàm. Có hay không có danh sách này tuỳ thuộc vào hàm cần định nghĩa;

- <*kiểu DL hàm*> kiểu dữ liệu mà hàm sẽ trả lại. Phần này bắt buộc phải được khai báo với mỗi hàm;

- <*thủ tục*> thân chương trình con. Trong đó câu lệnh <tên hàm> = <biểu thức> phải xuất hiện ít nhất một lần trong thủ tục. Câu lệnh này có tác dụng gán giá trị cho hàm.

Chú ý:

- Nếu không có từ khoá Public trước Function, hàm đó chỉ có tác dụng cục bộ: trong một module, trong một report hoặc trong một form.

- Khi có từ khoá Public trước Function, hàm sẽ có tác dụng toàn cục. Tức là có thể sử dụng bất kỳ nơi nào trên tệp Access đó. Tất nhiên, tất cả những gì khai báo là Public phải được khai báo trong phần **Decralations** của một Module nào đó.

Ví dụ:

* Thiết kế hàm tổng để tính tổng hai số:

```
Function Tong(a, b As Double) As Double
    Tong = a + b
End Function
```

* Thiết kế hàm kiểm tra tính nguyên tố của một số:

```
Function laNguyenTo(so As Integer) As Boolean
    Dim uoc As Integer
    laNguyenTo = True
    If so > 2 Then
        For uoc = 2 To Int(Sqr(so))
            If so Mod uoc = 0 Then
                laNguyenTo = False
                Exit For
            End If
            Next
    End If
End Function
```

5.2 Chương trình con dạng thủ tục

Cú pháp

[Public] [Private] **Sub** <*tên CTC*>([<*danh sách các tham số*>])
<*thủ tục*>
End Sub

Trong đó:

- **Sub, End Sub** là các từ khoá bắt buộc khai báo cấu trúc một chương trình con dạng thủ tục;

- <*tên CTC*> là tên gọi thủ tục định khai báo. Tên không được chứa dấu cách (space) và các ký tự đặc biệt;

- <*danh sách các tham số*> danh sách các tham số cần thiết cho thủ tục. Có hay không có danh sách này tuỳ thuộc vào thủ tục cần tạo

- <*thủ tục*> thân chương trình con.

Nếu không có từ khoá Public trước Sub, thủ tục đó chỉ có tác dụng cục bộ: trong một module, trong một report hoặc trong một form. Khi có từ khoá Public trước Sub, thủ tục sẽ có tác dụng toàn cục. Tức là có thể sử dụng bất kỳ nơi nào trên tệp Access đó. Tất nhiên, tất cả những gì khai báo là Public phải được khai báo trong phần Decralations của một Module nào đó.

Ví dụ:

***** Xây dựng thủ tục thực hiện tính tổng hai số:

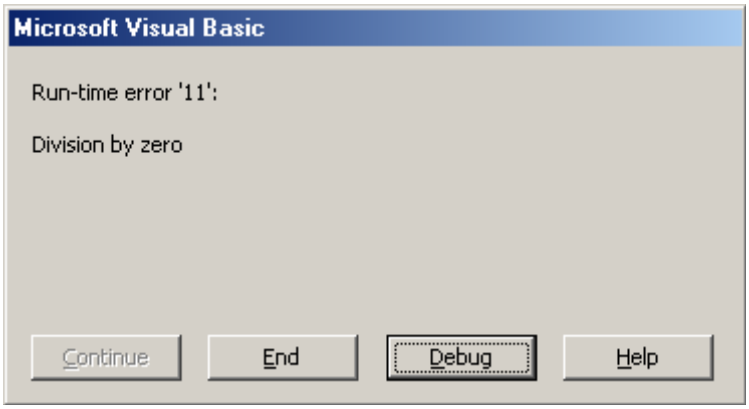
```
Sub tong2so(a, b As Double)
     tong = a + b      'tong là biến được khai báo
toàn cục
End Sub
```

6. Kỹ thuật xử lý lỗi

Xử lý lỗi là kỹ thuật rất quan trọng trong lập trình. Đã lập trình thì khó tránh khỏi lỗi (Errors). Có rất nhiều nguyên nhân gây ra lỗi; các nguyên nhân này có thể được lường trước hoặc không được lường trước. Kỹ thuật xử lý lỗi bao gồm các kỹ năng phát hiện và xử lý các tình huống khi chương trình gây lỗi.

6.1 Xử lý lỗi

Là việc xử lý khi đang lập trình gặp phải lỗi. Thông thường khi chạy thử chương trình trong lúc đang xây dựng phần mềm nếu gặp phải lỗi, sẽ xuất hiện hộp thoại thông báo lỗi có dạng:



Thông thường một hộp thoại thông báo lỗi gồm 2 thành phần:

- Thành phần báo lỗi bao gồm:

+ **Mã số lỗi** - Mỗi lỗi mà VBA có thể kiểm tra được đều có một mã số, được hiển thị ở dòng thông báo: *Run-time error 'mã số lỗi':* Ví dụ trên là : *Run-time error '11'*:

+ **Tên lỗi**. Ở ví dụ trên tên lỗi là: *Division by zero* - lỗi sai kiểu dữ liệu.

- Thành phần xử lý lỗi gồm 2 nút lệnh:

+ Nút End để dừng ngay chương trình, chuyển về chế độ **Design** - thiết kế bình thường;

+ Nút Debug để dừng chương trình chuyển về chế độ **Break** - sửa lỗi trực tiếp. Khi đó câu lệnh lỗi sẽ được tô bởi màu nền vàng cho phép người lập trình có thể sửa được mã chương trình.

Chú ý: Cửa sổ Immediate là công cụ hữu hiệu hỗ trợ việc dò lỗi bởi: hộp thoại này cho phép thực thi từng câu lệnh trên chế độ hội thoại.

6.2 Bẫy lỗi

a, Sử dụng lệnh On Error Resume Next

Sau khi sử dụng bẫy On Error Resume Next, từ chỗ đó trở đi, nếu chương trình gặp lỗi, nó sẽ bỏ qua (ignore) hoàn toàn. Điểm này tiện ở chỗ giúp chương trình EXE của ta tránh gặp lỗi thoát khỏi đột ngột như phân tích ở trên. Nhưng nó cũng bất lợi là khi khách hàng cho hay họ gặp những trường hợp lạ, không giải thích được (vì lỗi đã bị bỏ qua mà không ai để ý), thì ta cũng bí luôn, có thể không biết bắt đầu từ đâu để gỡ lỗi.

Do đó, trong lúc gỡ lỗi ta không nên dùng nó, nhưng trước khi giao cho khách hàng bạn nên cân nhắc kỹ có nên sử dụng trong các đoạn mã lệnh hay không.

Ví dụ: Sử dụng **On Error Resume Next** để bỏ qua lỗi:

```
Function A_chia_B(a, b As Double) As Double
    On Error Resume Next
    A_chia_B = Null
    A_chia_B = a / b
End Function
```

Trong chương trình con trên, nếu *b = 0*, lệnh *A_chia_B = a / b* sẽ gặp phải lỗi. Do có lời khai báo On Error Resume Next nên lệnh lỗi này được bỏ qua (không thực hiện). Tức là giá trị hàm là Null.

b, Sử dụng câu lệnh On Error Goto <nhãn>

Khi một thủ tục được đặt câu lệnh này, nếu gặp phải một lỗi nào đó, VBA sẽ chuyển thẳng việc thực hiện đến *<nhãn>* đã chỉ định. Thông thường các lệnh tiếp theo của *<nhãn>* là xử lý các tình huống lỗi.

Ví dụ: Sử dụng phương pháp **On Error Goto <*nhãn*>** để bẫy lỗi:

```
Function A_chia_B(a, b As Double) As Double
     On Error GoTo Loi
     A_chia_B = a / b
     Msgbox " Ok! "
     Loi:
     If Err.Number = 11 Then
          MsgBox "Lỗi chia cho 0 !"
     End If
End Function
```

Trong chương trình con trên, trong trường hợp *b = 0* câu lệnh *A_chia_B = a / b* sẽ gây ra lỗi. Theo như khai báo **On Error Goto Loi** ban đầu, VBA sẽ bỏ qua tất cả các lệnh sau lệnh lỗi và chuyển thẳng tới các lệnh sau nhãn **Loi:** Ở đây là lệnh kiểm tra lỗi. Nếu *Mã lỗi = 11* kết luận ngay một thông báo lỗi tiếng Việt. *Lỗi chia cho 0 !*

Phương pháp này cũng được dùng phổ biến cả trong quá trình xây dựng để phát hiện lỗi, cũng như trong phần mềm đã đóng gói gửi đến khách hàng. Mỗi khi gặp lỗi sẽ được thông báo nguyên nhân gây ra lỗi bằng tiếng Việt (chẳng hạn) mà vẫn không ảnh hưởng đến hoạt động khác của phần mềm.

Trong phương pháp này, người lập trình nên khai thác tối đa đối tượng **Err** - đối tượng mang những thông tin về lỗi đang xảy ra, cụ thể:

Hành động	Kết quả
Err.Description	Mô tả tên lỗi
Err.Number	Đưa ra mã lỗi