

## Lời nói đầu

Tự động hóa trong tất cả lĩnh vực hiện đang được xã hội quan tâm đặc biệt bởi nhờ nó năng suất lao động được nâng cao, chất lượng sản phẩm ổn định và tốt hơn, nhiều ý tưởng mới có cơ hội trở thành hiện thực. Tự động hóa công tác thiết kế công trình giao thông cũng không nằm ngoài quy luật chung đó, hiện nay, hầu hết các công ty trong lĩnh vực tư vấn thiết kế công trình giao thông đều rất chú trọng thực hiện tự động hóa công tác thiết kế trong công ty của mình. Điều này được thể hiện rõ nét trong việc đầu tư của các công ty (mua sắm máy tính, phần mềm và đào tạo nhân lực) cũng như triển khai tự động hóa thiết kế rất nhiều công trình trong thực tế.

Với sự đa dạng của mình, các bài toán trong công tác thiết kế luôn đòi hỏi sự linh hoạt của công tác tự động hóa. Chính vì vậy, để phân nào đáp ứng được yêu cầu cấp bách từ thực tế sản xuất, nội dung cuốn giáo trình này đề cập đến tất cả các vấn đề cơ bản nhất của việc thực hiện tự động hóa thiết kế công trình giao thông cũng như phương pháp để nâng cao mức độ tự động hóa cho phù hợp với từng yêu cầu chuyên biệt xuất hiện trong quá trình thiết kế.

Nội dung của giáo trình này là sự đúc kết kinh nghiệm giảng dạy môn Tự động hóa thiết kế cầu đường cho sinh viên ngành xây dựng công trình giao thông và quá trình tham gia thực hiện tự động hóa công tác thiết kế ngoài sản xuất của các tác giả cũng như cập nhật mới nhất những công nghệ chủ chốt phục vụ cho việc tự động hóa. Hơn nữa, nội dung chính tập trung vào những thành phần cốt lõi phục vụ cho mục đích tự động hóa thiết kế cầu đường, cùng với những nội dung mang tính gợi mở và định hướng cho từng chuyên ngành, khiến cho cuốn giáo trình này hoàn toàn phù hợp với định hướng đào tạo theo tín chỉ của Nhà trường.

Chúng tôi xin chân thành cảm ơn sự đóng góp ý kiến của các đồng nghiệp trong quá trình hoàn thiện cuốn giáo trình này.

Với tốc độ phát triển rất nhanh của công nghệ như hiện nay thì chắc chắn rằng trong thời gian tới, nhiều vấn đề liên quan đến việc thực hiện tự động hóa thiết kế sẽ phải thay đổi, và chúng tôi hy vọng rằng, cùng với các ý kiến đóng góp của bạn đọc và sự cập nhật kiến thức của bản thân, thì lần xuất bản sau của cuốn sách này sẽ hoàn thiện hơn nữa, sẽ đáp ứng tốt hơn nữa yêu cầu của bạn đọc.

Hà Nội, ngày 01 tháng 06 năm 2007

**Các tác giả.**



## **PHẦN I: MỞ ĐẦU ..... 1**

1. Tổng quan về thiết kế và tự động hóa thiết kế công trình giao thông.....	1
2. Đôi nét về các phần mềm dùng cho thiết kế công trình giao thông.....	3
3. Lựa chọn phần mềm dùng cho thiết kế công trình giao thông.....	4
4. Chuyên biệt hóa phần mềm.....	6
5. Kết chương.....	11

## **PHẦN II: LẬP TRÌNH TRÊN ỨNG DỤNG NỀN ..... 12**

### **CHƯƠNG I: KHÁI NIỆM..... 12**

### **CHƯƠNG II: TỔNG QUAN VỀ VBA ..... 19**

1. Đặc điểm của VBA.....	19
2. Trình tự xây dựng một dự án bằng VBA.....	19
3. Cấu trúc của một dự án VBA.....	20
4. Môi trường phát triển tích hợp VBA IDE.....	21
5. Ví dụ đầu tiên với VBA.....	23

### **CHƯƠNG III: CƠ BẢN VỀ NGÔN NGỮ LẬP TRÌNH VISUAL BASIC ..... 25**

1. Những quy định về cú pháp.....	25
2. Các trợ giúp về cú pháp trong quá trình viết mã lệnh.....	25
3. Tính năng gọi nhớ và tự hoàn thiện mã lệnh.....	26
4. Từ khoá trong VB.....	27
5. Các kiểu dữ liệu cơ bản.....	28
5.1. Kiểu logic (boolean).....	29
5.2. Kiểu số nguyên.....	29
5.3. Kiểu số thực.....	29
5.4. Kiểu mảng (array).....	29
5.5. Kiểu chuỗi (String).....	31
5.6. Kiểu thời gian (Date).....	32
5.7. Kiểu Variant.....	32
5.8. Kiểu tự định nghĩa (user-defined type).....	33
5.9. Kiểu lớp (Class).....	34
6. Khai báo biến trong VB.....	35
6.1. Khai báo hằng số.....	38
6.2. Khai báo biến.....	38
6.3. Khai báo kiểu tự định nghĩa.....	38
6.4. Khai báo mảng tĩnh.....	39
6.5. Khai báo mảng động.....	39
6.6. Khai báo, tạo và làm việc với biến đối tượng.....	40
7. Các toán tử và hàm thông dụng.....	40
7.1. Các toán tử.....	40
7.2. Các hàm toán học.....	41
7.3. Các hàm chuyển đổi dữ liệu.....	41
7.4. Các hàm xử lý chuỗi.....	43
8. Các cấu trúc điều khiển.....	44
8.1. Cấu trúc điều kiện.....	44
8.2. Cấu trúc lựa chọn.....	46
8.3. Vòng lặp xác định.....	47

8.3.1. Vòng lặp theo biến đếm .....	47
8.3.2. Lặp trong một tập hợp.....	49
8.4. Vòng lặp không xác định .....	50
9. Chương trình con .....	51
9.1. Hàm (Function).....	52
9.2. Thủ tục (Sub) .....	52
9.3. Truyền tham số cho chương trình con.....	52
9.3.1. Truyền tham số theo tham chiếu .....	53
9.3.2. Truyền tham số theo tham trị .....	54
9.3.3. Tham số tùy chọn.....	54
9.3.4. Danh sách tham số với số lượng tham số tùy ý.....	55
9.3.5. Hàm có giá trị trả về là kiểu mảng .....	55
9.4. Biến trong chương trình con .....	56
9.5. Cách thức gọi chương trình con.....	58
9.6. Thoát khỏi chương trình con.....	59
10. Tổ chức các chương trình con theo hệ thống các mô-đun chuẩn .....	59
11. Làm việc với UserForm và các thành phần điều khiển .....	60
11.1. Các vấn đề chung .....	60
11.1.1. Tạo UserForm và các thành phần điều khiển trong VBA IDE .....	63
11.1.2. Các thuộc tính của UserForm và các thành phần điều khiển.....	64
11.1.3. Các phương thức của UserForm và các thành phần điều khiển.....	66
11.1.4. Các sự kiện trên giao diện.....	66
11.1.5. Ví dụ.....	67
11.2. Làm việc với UserForm .....	68
11.3. Các điều khiển thông dụng.....	69
12. Các hộp thoại thông dụng.....	76
12.1. Hộp thông điệp (Message Box – MsgBox).....	76
12.2. Hộp nhập dữ liệu (Input Box – InputBox).....	77
12.3. Hộp thoại dựa trên điều khiển Common Dialog .....	78
13. Lập trình xử lý tập tin.....	80
13.1. Các hình thức truy cập tập tin .....	81
13.2. Xử lý dữ liệu trong tập tin với các hàm I/O: .....	82
13.2.1. Mở tập tin:.....	82
13.2.2. Đọc dữ liệu từ tập tin: .....	82
13.2.3. Ghi dữ liệu vào tập tin: .....	84
13.2.4. Đóng tập tin.....	86
13.3. Xử lý dữ liệu trong tập tin theo mô hình FSO (File System Object) .....	86
13.3.1. Tạo tập tin mới .....	88
13.3.2. Mở tập tin đã có để thao tác.....	89
14. Gỡ rối và bẫy lỗi trong VBAIDE .....	90
14.1. Phân loại lỗi trong lập trình.....	90
14.2. Gỡ rối trong lập trình .....	91
14.2.1. Phát hiện lỗi lúc thực thi .....	91
14.2.2. Các phương pháp thực thi mã lệnh .....	92
14.2.3. Cửa sổ trợ giúp gỡ rối .....	93
14.3. Bẫy lỗi trong VBAIDE.....	95
14.3.1. Câu lệnh On Error .....	95
14.3.2. Đối tượng Err .....	96
14.3.3. Hàm Error .....	97
<b>CHƯƠNG IV: LẬP TRÌNH TRÊN MICROSOFT EXCEL .....</b>	<b>99</b>
1. Tổng quan về Microsoft Excel .....	99
1.1. Khả năng của Excel.....	99
1.2. Giao diện của Excel .....	99
1.3. Khả năng mở rộng của Excel .....	100
2. Macro .....	100

2.1. Macro là gì?.....	101
2.2. Tạo Macro .....	101
2.2.1. Tạo Macro theo kịch bản.....	101
2.2.2. Tạo Macro sử dụng VBA.....	104
2.3. Quản lý Macro.....	104
2.4. Sử dụng Macro .....	105
2.4.1. Thực thi Macro bằng phím tắt.....	106
2.4.2. Thực thi Macro thông qua trình quản lý Macro .....	106
2.4.3. Thực thi Macro trực tiếp từ VBAIDE .....	106
2.5. Hiệu chỉnh Macro.....	107
2.6. Vấn đề an toàn khi sử dụng Macro.....	107
3. Xây dựng hàm mới trong Excel.....	107
3.1. Khái niệm về hàm trong Excel.....	107
3.2. Tạo hàm mới bằng VBA .....	108
3.2.1. Tại sao phải dùng hàm?.....	108
3.2.2. Cấu trúc hàm .....	109
3.2.3. Tạo hàm mới .....	109
3.3. Hàm trả về lỗi.....	111
4. Add-in và Phân phối các ứng dụng mở rộng.....	113
4.1. Khái niệm về Add-In.....	114
4.2. Trình quản lý Add-In.....	114
4.3. Tạo Add-In.....	115
4.4. Phân phối và Cài đặt Add-In .....	117
5. Hệ thống các đối tượng trong Excel.....	117
5.1. Mô hình đối tượng trong Excel .....	117
5.2. Một số đối tượng cơ bản trong Excel .....	119
5.2.1. Đối tượng Application.....	119
5.2.2. Đối tượng Workbook .....	123
5.2.3. Đối tượng Window.....	126
5.2.4. Đối tượng Worksheet .....	128
5.2.5. Đối tượng Range .....	131
5.2.6. Tập đối tượng Cells.....	135
6. Sự kiện của các đối tượng trong Excel.....	137
6.1. Tạo bộ xử lý sự kiện cho một sự kiện .....	138
6.2. Sự kiện trong Workbook .....	139
6.3. Sự kiện trong Worksheet.....	141
6.4. Sự kiện trong UserForm.....	143
6.5. Sự kiện không gắn với đối tượng .....	144
7. Các thao tác cơ bản trong Excel .....	145
7.1. Điều khiển Excel .....	146
7.1.1. Thoát khỏi Excel .....	146
7.1.2. Khóa tương tác người dùng.....	147
7.1.3. Thao tác với cửa sổ .....	147
7.1.4. Khởi động Excel từ chương trình khác .....	148
7.2. Làm việc với Workbook.....	150
7.2.1. Tạo mới, mở, lưu và đóng workbook.....	150
7.3. Làm việc với Worksheet .....	151
7.3.1. Tạo mới, xoá và đổi tên worksheet .....	151
7.4. Làm việc với Range và Cells.....	152
7.4.1. Duyệt qua từng ô trong vùng dữ liệu.....	152
7.4.2. Duyệt qua từng ô trong vùng dữ liệu theo hàng và cột .....	152
7.4.3. Vùng có chứa dữ liệu – Thuộc tính UsedRange.....	153
7.5. Làm việc với biểu đồ.....	153
7.5.1. Tạo mới biểu đồ .....	154
7.5.2. Thêm một chuỗi số liệu vào biểu đồ đã có.....	155
7.6. Sử dụng các hàm có sẵn trong Excel.....	157

8. Giao diện người dùng.....	157
8.1. Điều khiển nhúng trong Worksheet .....	157
8.1.1. Điều khiển Spin Button.....	158
8.1.2. Điều khiển ComboBox .....	159
8.1.3. Điều khiển Command Button .....	160
8.2. Các hộp thoại thông dụng .....	161
8.2.1. Hộp thoại InputBox của Excel – Hàm InputBox .....	161
8.2.2. Hộp thoại Open – Hàm GetOpenFilename .....	163
8.2.3. Hộp thoại Save As – Hàm GetSaveAsFilename .....	165
8.2.4. Hộp thoại chọn thư mục – Đối tượng FileDialog .....	166
8.2.5. Các hộp thoại mặc định trong Excel – Tập đối tượng Dialogs .....	166
8.2.6. Thực thi mục trình đơn Excel từ VBA.....	168
8.3. Hộp thoại tùy biến – UserForm.....	169
8.3.1. Tạo mới UserForm.....	169
8.3.2. Hiển thị UserForm .....	170
8.3.3. Các điều khiển trên UserForm .....	171
8.4. Thao tác trên thanh trình đơn .....	172
8.4.1. Cấu trúc của hệ thống thanh trình đơn .....	173
8.4.2. Tạo trình đơn tùy biến.....	174
8.4.3. Xoá trình đơn tùy biến .....	177
8.4.4. Gán phím tắt cho Menu Item .....	178
<b>CHƯƠNG V: LẬP TRÌNH TRÊN AUTOCAD.....</b>	<b>181</b>
1. Tổng quan về AutoCAD .....	181
1.1. Khả năng của AutoCAD .....	181
1.2. Giao diện của AutoCAD .....	182
1.3. Khả năng mở rộng của AutoCAD.....	183
2. Quản lý dự án VBA trong AutoCAD .....	184
2.1. Dự án VBA trong AutoCAD.....	184
2.2. Trình quản lý dự án VBA.....	185
2.2.1. Tạo mới, Mở và Lưu dự án VBA.....	186
2.2.2. Nhúng và tách dự án VBA .....	187
2.3. Quản lý dự án VBA từ dòng lệnh .....	188
3. Macro .....	188
3.1. Khái niệm Macro trong AutoCAD.....	188
3.2. Tạo mới và Hiệu chỉnh Macro .....	189
3.3. Thực thi Macro.....	190
3.4. Định nghĩa lệnh mới bằng AutoLISP.....	191
3.4.1. Tạo dự án mới .....	191
3.4.2. Tạo và thử nghiệm Macro HelloWorld.....	192
3.4.3. Tạo lệnh mới bằng AutoLISP .....	193
4. Hệ thống đối tượng trong AutoCAD .....	193
4.1. Mô hình đối tượng trong AutoCAD.....	193
4.2. Một số đối tượng chính trong AutoCAD .....	195
4.2.1. Đối tượng Application .....	195
4.2.2. Đối tượng Document.....	196
4.2.3. Tập đối tượng.....	198
4.2.4. Đối tượng phi hình học .....	198
4.2.5. Đối tượng hình học .....	199
5. Các thao tác cơ bản trong AutoCAD .....	200
5.1. Điều khiển AutoCAD.....	200
5.1.1. Tạo mới, Mở, Lưu và Đóng bản vẽ.....	200
5.1.2. Khởi động và thoát khỏi chương trình AutoCAD.....	203
5.1.3. Sử dụng các lệnh sẵn có của AutoCAD .....	205
5.1.4. Thu phóng màn hình bản vẽ (zoom).....	205
5.1.5. Nhập dữ liệu người dùng từ dòng lệnh của AutoCAD .....	207

5.1.6. Thiết lập biến hệ thống.....	214
5.2. Tạo mới đối tượng hình học.....	217
5.2.1. Xác định nơi chứa đối tượng.....	217
5.2.2. Khai báo và tạo đối tượng hình học.....	218
5.2.3. Tạo đối tượng Point.....	219
5.2.4. Tạo đối tượng dạng đường thẳng.....	220
5.2.5. Tạo đối tượng dạng đường cong.....	223
5.2.6. Tạo đối tượng văn bản.....	225
5.3. Làm việc với đối tượng SelectionSet.....	227
5.3.1. Khai báo và khởi tạo đối tượng SelectionSet.....	228
5.3.2. Thêm đối tượng hình học vào một SelectionSet.....	228
5.3.3. Thao tác với các đối tượng trong SelectionSet.....	234
5.3.4. Định nghĩa bộ lọc đối tượng cho SelectionSet.....	234
5.3.5. Loại bỏ đối tượng hình học ra khỏi SelectionSet.....	236
5.4. Hiệu chỉnh đối tượng hình học.....	237
5.4.1. Hiệu chỉnh đối tượng sử dụng các phương thức.....	238
5.4.2. Hiệu chỉnh đối tượng sử dụng các thuộc tính.....	245
5.4.3. Hiệu chỉnh đường đa tuyến.....	249
5.4.4. Hiệu chỉnh văn bản đơn.....	251
5.5. Làm việc với lớp (Layer).....	253
5.5.1. Tạo lớp mới.....	254
5.5.2. Truy xuất và thay đổi tên một lớp đã có.....	255
5.5.3. Thiết lập lớp hiện hành.....	255
5.5.4. Thiết lập các chế độ hiển thị của lớp.....	255
5.5.5. Xoá lớp.....	257
5.6. Thao tác với kiểu đường – Linetype.....	257
5.6.1. Tải kiểu đường vào AutoCAD.....	257
5.6.2. Truy xuất và đổi tên kiểu đường.....	258
5.6.3. Thiết lập kiểu đường hiện hành.....	259
5.6.4. Xoá kiểu đường đã có.....	259
5.7. Thao tác với đường kích thước – Dimension.....	259
5.7.1. Kiểu đường kích thước – DimensionStyle.....	260
5.7.2. Tạo đường kích thước.....	262
5.7.3. Định dạng đường kích thước.....	267
5.8. Thao tác với dữ liệu mở rộng – XData.....	268
5.8.1. Gán dữ liệu mở rộng.....	268
5.8.2. Đọc dữ liệu mở rộng.....	269
6. Giao diện người dùng.....	270
6.1. Thao tác với thanh trình đơn.....	270
6.1.1. Cấu trúc của hệ thống thanh trình đơn.....	270
6.1.2. Tạo trình đơn.....	272
6.1.3. Xoá thanh trình đơn.....	274

---

**PHẦN III: TÀI LIỆU THAM KHẢO..... 276**





## PHẦN I: MỞ ĐẦU

### 1. Tổng quan về thiết kế và tự động hóa thiết kế công trình giao thông

Công tác thiết kế luôn có một vị trí quan trọng từ khi lập dự án cho đến khi thi công, hoàn thành và đưa công trình vào sử dụng. Từ trước đến nay, công tác khảo sát thiết kế được biết đến như một quá trình gồm nhiều công đoạn khác nhau, mà mục đích cuối cùng là xác lập cấu tạo của công trình, cách thức thi công chủ đạo để tạo ra công trình trên thực địa và phương pháp khai thác công trình một cách hiệu quả nhất. Kết quả của công tác thiết kế được thể hiện dưới dạng hồ sơ thiết kế, nghĩa là quá trình thiết kế nhằm đến việc tạo ra một bộ hồ sơ thiết kế, mà trong đó nó mô tả một cách đầy đủ toàn bộ mục đích của quá trình thiết kế. Thông thường hồ sơ thiết kế bao gồm những thành phần cơ bản như sau:

- ◆ Bản thuyết minh: nơi thể hiện những cơ sở cho công tác thiết kế, lập luận của người thiết kế và giải thích những vấn đề cơ bản của phương án thiết kế.
- ◆ Các loại bảng tính, bảng thống kê: nơi trình bày các kết quả tính toán trong quá trình thiết kế, là cơ sở cho việc lập bản vẽ và xác định chi phí đầu tư cho công trình.
- ◆ Bản vẽ: nơi thể hiện chi tiết nhất cấu tạo của công trình cũng như phương pháp chủ đạo để thi công công trình.
- ◆ Dự toán: nơi thể hiện cách thức xác định tổng mức đầu tư cho công trình.

Mức độ chi tiết của những thành phần trong hồ sơ thiết kế phụ thuộc vào yêu cầu trong từng giai đoạn của quá trình đầu tư cho công trình. Ví dụ giai đoạn lập bản vẽ thi công đòi hỏi mức độ chi tiết cao nhất.

Nếu xem xét kỹ hơn bên trong của hồ sơ thiết kế công trình giao thông thì ai cũng nhận thấy rằng chúng có mối liên hệ chặt chẽ với nhau theo một quan hệ logic khá rõ ràng, ví dụ các kích thước hình học trong bản vẽ sẽ phải phù hợp với kết quả tính toán được trình bày trong các bảng tính. Điều này nói lên rằng, khi mô tả mối liên hệ trên thành một chuỗi các lệnh thì ta đã có trong tay thành phần cơ bản nhất của tự động hóa thiết kế công trình giao thông. Vấn đề còn lại là tìm kiếm giải pháp thích hợp để thực hiện tự động hóa.

Tự động hóa một công việc được hiểu là công việc đó được thực hiện tự động hoàn toàn hay một phần nhờ có sự trợ giúp của các thiết bị. Ví dụ như quá trình chế tạo xe hơi được tự động hóa nhờ hệ thống robot trong các dây chuyền sản xuất. Trong lĩnh vực thiết kế công trình giao thông, do sản phẩm của công tác này là hồ sơ thiết kế, cho nên thiết bị trợ giúp phù hợp là các hệ thống có khả năng tạo văn bản, tính toán kết cấu, vẽ các đối tượng hình học, dựng mô hình...

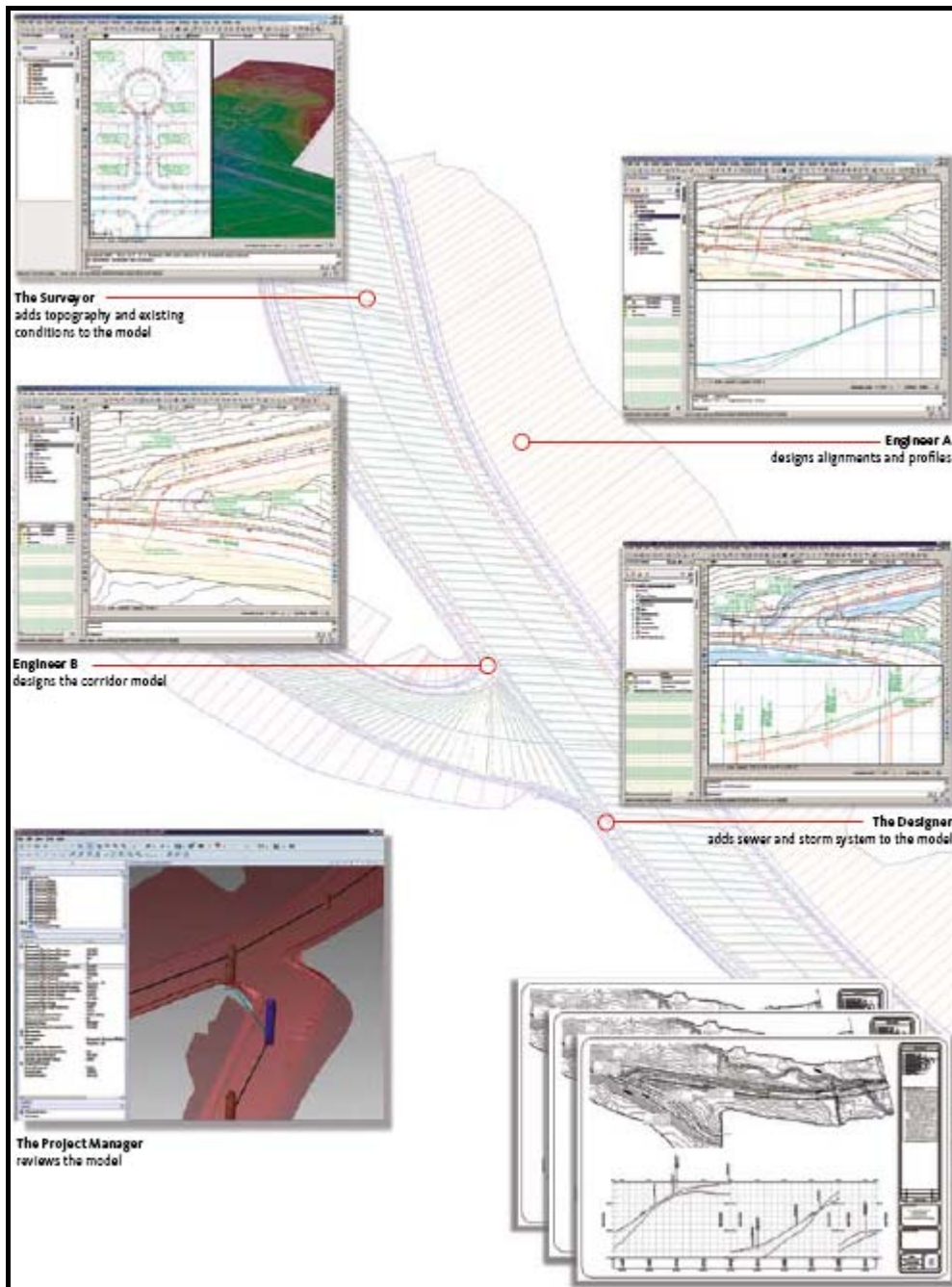
Hệ thống thông tin, bao gồm phần cứng (máy tính, máy in, máy quét...) và phần mềm (các chương trình ứng dụng), đã và đang được triển khai rộng rãi trong khắp các công ty tư vấn thiết kế công trình giao thông bởi chúng có những đặc điểm rất phù hợp cho việc lập hồ sơ thiết kế công trình:

- ◆ Máy tính cùng với các phần mềm chạy trên chúng cho phép thực hiện nhiều công việc khác nhau như: phân tích kết cấu, vẽ đối tượng hình học, tạo văn bản, dựng mô hình...
- ◆ Tốc độ tính toán nhanh, điều này cho phép đưa ra nhiều hơn một phương án thiết kế với thời gian có thể chấp nhận được.
- ◆ Khả năng lưu trữ và tận dụng lại dữ liệu đạt hiệu quả rất cao, điều này cho phép người thiết kế có thể tận dụng lại tối đa dữ liệu đã có từ trước. Ví dụ, với hệ thống các bản vẽ in trên giấy, việc tận dụng lại đạt hiệu quả rất thấp, hầu như chỉ ở mức tham khảo thông tin,

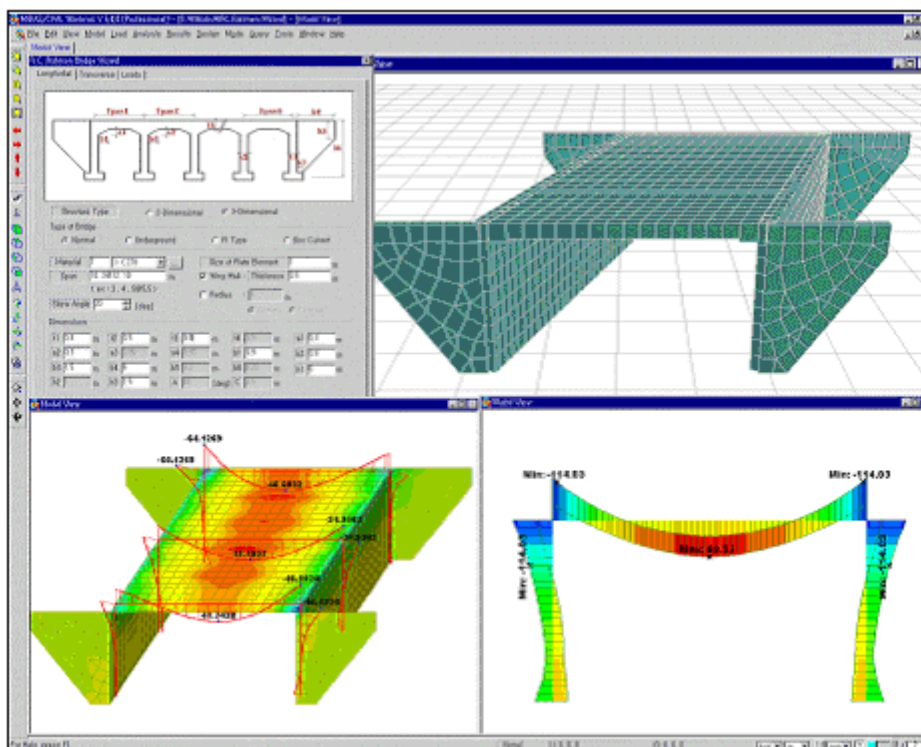
## GIÁO TRÌNH TỰ ĐỘNG HOÁ THIẾT KẾ CẦU ĐƯỜNG

trong khi đó, nếu như cũng các bản vẽ này được lưu trữ trong máy tính, ngoài việc cho phép tham khảo tương tự như bản vẽ in trên giấy, nó còn cho phép tận dụng lại chính các thành phần trong bản vẽ đó để chỉnh sửa, kế thừa, và kết quả ta sẽ có được một bản vẽ mới từ những dữ liệu cũ.

Có thể nói rằng mức độ tự động hóa thiết kế công trình hiện nay đang ở nhiều cấp độ khác nhau, tùy theo từng công việc cụ thể, điều này được thể hiện rõ trong cách thức tạo ra từng thành phần trong hồ sơ thiết kế. Ví dụ, trong thiết kế cầu, phần phân tích kết cấu có mức độ tự động hóa rất cao, nhưng việc tạo bản vẽ lại có mức độ tự động hóa thấp hơn nhiều. Tuy vậy, xu hướng nâng cao mức độ tự động hóa đang ngày càng rõ nét bởi sự phát triển rất mạnh của các phần mềm chuyên dụng, chúng đang là công cụ hỗ trợ không thể thiếu cho các kỹ sư thiết kế, đồng thời là thành phần chủ chốt cho quá trình tự động hóa. Nhờ chúng mà việc phân tích kết cấu công trình trở nên nhanh chóng và chính xác, nhờ chúng mà việc đưa ra các phương án thiết kế của tuyến đường cũng như việc tạo mô hình ba chiều động trở thành hiện thực.



Hình I-1: Tự động hóa thiết kế hình học đường ô tô với Civil 3D 2008



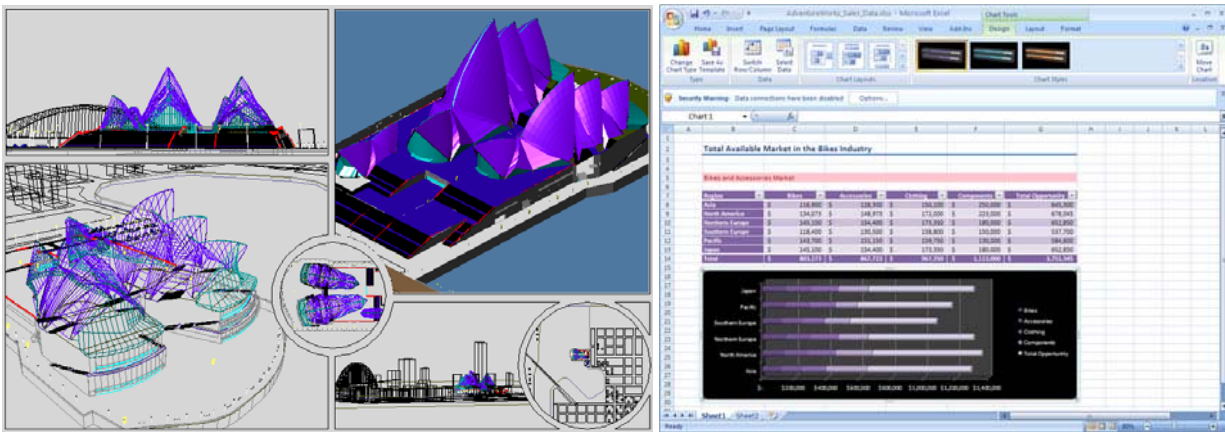
Hình I-2: Tự động hóa phân tích kết cấu với Midas Civil

## 2. Đôi nét về các phần mềm dùng cho thiết kế công trình giao thông

Các phần mềm dùng trong thiết kế công trình nói chung rất đa dạng và hỗ trợ hầu hết các công đoạn trong quá trình thiết kế. Ngay từ công đoạn khảo sát địa hình, toàn bộ quá trình từ xử lý dữ liệu (bình sai, chuyển đổi định dạng) đến dựng mô hình bề mặt đều đã được tự động hóa ở mức cao, hầu hết các nội dung liên quan đến xử lý số liệu khảo sát đều được tự động thực hiện như: vẽ đường đồng mức, phân tích độ dốc bề mặt, xác định đường tự thủy, xác định lưu vực, vẽ mặt cắt và dựng mô hình ba chiều.

Dựa vào công năng của các phần mềm có thể chia chúng làm hai nhóm:

- ◆ Nhóm các phần mềm đa năng: là những phần mềm có thể dùng cho nhiều mục đích khác nhau, đại diện cho nhóm này là AutoCAD và Excel, ta có thể sử dụng chúng trong hầu hết các giai đoạn của quá trình tạo hồ sơ thiết kế. Tuy nhiên, để có thể sử dụng đa năng, các phần mềm này được thiết kế không tập trung vào một lĩnh vực cụ thể nào, khiến cho mức độ tự động hóa cho từng công việc không được cao khi thực hiện trực tiếp trên các phần mềm này. Ta có thể dùng AutoCAD để tạo các bản vẽ kỹ thuật cho ngành cơ khí cũng như công trình, bởi nguyên tắc tạo bản vẽ trong AutoCAD là “lắp ghép” từ những đối tượng hình học cơ bản. Với Excel, ta có thể dùng để lập dự toán hay tạo bảng tính duyệt kết cấu, bởi mỗi ô trong bảng tính của nó đều có thể nhận bất cứ nội dung nào.



**Hình I-3: AutoCAD và Excel**

◆ Nhóm các phần mềm chuyên dụng: là các phần mềm chỉ dùng được cho một mục đích cụ thể nào đó. Bởi đích nhắm đến của chúng là rõ ràng cho nên mức độ tự động hóa là rất cao. Ví dụ trong phân tích kết cấu, sau khi nhập xong số liệu, phần mềm phân tích kết cấu sẽ tự động hoàn toàn trong việc tính và xuất kết quả. Bởi sự đa dạng của các bài toán thiết kế, cho nên các phần mềm loại này cũng rất đa dạng về chủng loại và nguồn gốc, chúng có thể được tạo ra từ những công ty sản xuất phần mềm chuyên nghiệp như Hải Hòa, AutoDesk, MIDAS IT, ... hay từ chính những công ty tư vấn thiết kế, và thậm chí từ chính những kỹ sư thiết kế. Cũng bởi tính đa dạng này mà việc lựa chọn để tìm được một phần mềm phù hợp đôi khi là một bài toán khó đối với người sử dụng. Dựa trên mức độ phổ biến trong sử dụng, có thể kể ra một số phần mềm chuyên dụng sau:

- Trong lĩnh vực phân tích kết cấu: MIDAS/Civil, RM, SAP, ANSYS, LUSAS, ABAQUS.
- Trong lĩnh vực địa kỹ thuật: Geo-Slope, Plaxis, MIDAS GTS.
- Trong lĩnh vực địa hình, bản đồ: Land Desktop, Topo, MapInfo, CAD Overlay.
- Trong lĩnh vực thiết kế hình học đường ô tô: Nova-TDN, Civil 3D.

Do công trình giao thông luôn phụ thuộc vào rất nhiều yếu tố xung quanh nó, cho nên quá trình thiết kế luôn gặp phải những bài toán riêng, đặc biệt và không thể khái quát được. Những bài toán này hầu như không có lời giải tổng quát, và cũng bởi điều này khiến cho không có một phần mềm chuyên dụng nào có thể giải quyết được mọi vấn đề, nhất là trong thiết kế đường ô tô. Bên cạnh đó, do có sự khác nhau trong cách trình bày và thể hiện bản vẽ, nên thông thường các phần mềm chuyên dụng chỉ có thể đáp ứng việc tạo bản vẽ ở mức cơ bản, còn việc bổ sung thêm chi tiết để hoàn thiện bản vẽ thường được làm thủ công. Những nhược điểm này của các phần mềm chuyên dụng lại là điều kiện cho sự ra đời các phần mềm dạng Add-in<sup>1</sup>, chúng thường được phát triển bởi các kỹ sư cầu đường trong công ty tư vấn thiết kế công trình giao thông và chạy cùng với các phần mềm chính, chúng tác động trực tiếp lên kết quả do phần mềm chính tạo ra với mục đích là hoàn thiện chúng theo yêu cầu riêng của chính công ty đó.

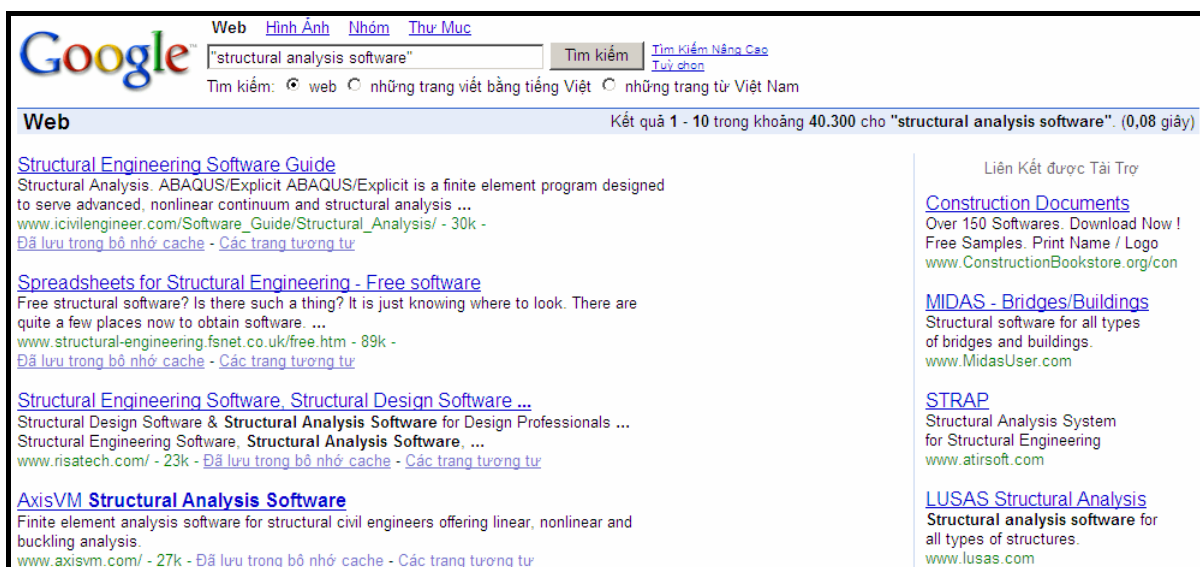
### 3. Lựa chọn phần mềm dùng cho thiết kế công trình giao thông

Với sự đa dạng về chủng loại và xuất xứ của các phần mềm chuyên dụng, khiến cho việc chọn mua phần mềm gặp nhiều khó khăn, nhất là đối với những đơn vị ít kinh nghiệm trong việc

<sup>1</sup> **Add-in**: đây là các chương trình dạng phụ trợ hoặc tiện ích được thiết kế để cùng hoạt động với chương trình chính. Mục đích dùng để mở rộng các khả năng cho chương trình chính. Các chương trình dạng Add-in này có thể do chính người dùng tạo ra bằng nhiều loại công cụ khác nhau. Không phải chương trình chính nào cũng chấp nhận Add-in, AutoCAD, MS.Office là hai phần mềm cho phép sử dụng Add-in điển hình.

triển khai các hệ thống phần mềm. Do đó, để trang bị được phần mềm phù hợp với công việc của mình cần phải thực hiện một số công việc chính sau:

- ◆ Chuẩn bị về nhân lực: để khai thác hiệu quả phần mềm, nhất là các phần mềm chuyên dụng, cần có nhân lực đáp ứng được cả hai yêu cầu:
  - Có kiến thức tin học cơ bản: sử dụng tốt hệ điều hành Windows (hoặc tương đương), in ấn, tìm kiếm tài liệu trên Internet.
  - Có kiến thức chuyên môn phù hợp.
- ◆ Phân tích công việc cần tự động hóa để xác định rõ các yêu cầu cần được thỏa mãn khi triển khai ứng dụng phần mềm. Ví dụ, để tự động hóa công tác thiết kế kết cấu, những yêu cầu sau cần được thỏa mãn:
  - Tính được nội lực và chuyển vị của kết cấu dưới tác dụng của các loại tải trọng (cần nêu cụ thể, ví dụ như các trường hợp tổ hợp tải trọng).
  - Đưa ra được mô tả về phân bố ứng suất tại một số vị trí (cần nêu cụ thể, ví dụ tại các nơi có cấu tạo hình học thay đổi đột ngột).
  - Có thể tính duyệt được mặt cắt.
  - Có thể tạo bản vẽ (cần nêu cụ thể mức độ chi tiết của bản vẽ) và hỗ trợ in ra máy in.
  - Có thể kết nối dữ liệu với các phần mềm khác (cần chỉ rõ định dạng kết nối, ví dụ yêu cầu nhập/xuất cấu tạo hình học của kết cấu từ/sang định dạng \*.DXF).
  - Có thể thêm các tính năng mới cho phần mềm bằng các công cụ dạng Add-in (yêu cầu này có thể không bắt buộc phải có).
- ◆ Tìm hiểu, càng nhiều càng tốt, các phần mềm chuyên dụng mà có thể đáp ứng được những yêu cầu trên. Có nhiều cách để thu thập thông tin:
  - Kinh nghiệm của các đơn vị, cá nhân đã sử dụng.
  - Giới thiệu từ nhà sản xuất phần mềm về tính năng, giá cả và chế độ hỗ trợ trong quá trình dùng sản phẩm của họ.
  - Đánh giá phần mềm của các tạp chí chuyên ngành.
  - Tìm thông tin liên quan trên Internet.



Hình I-4: Tìm kiếm thông tin trên Internet với Google.com

- Sử dụng phiên bản dùng thử miễn phí của phần mềm để tự kiểm chứng.

- ◆ Đàm phán với nhà cung cấp phần mềm để tìm ra một giải pháp hợp lý nhất trước khi quyết định mua sản phẩm.

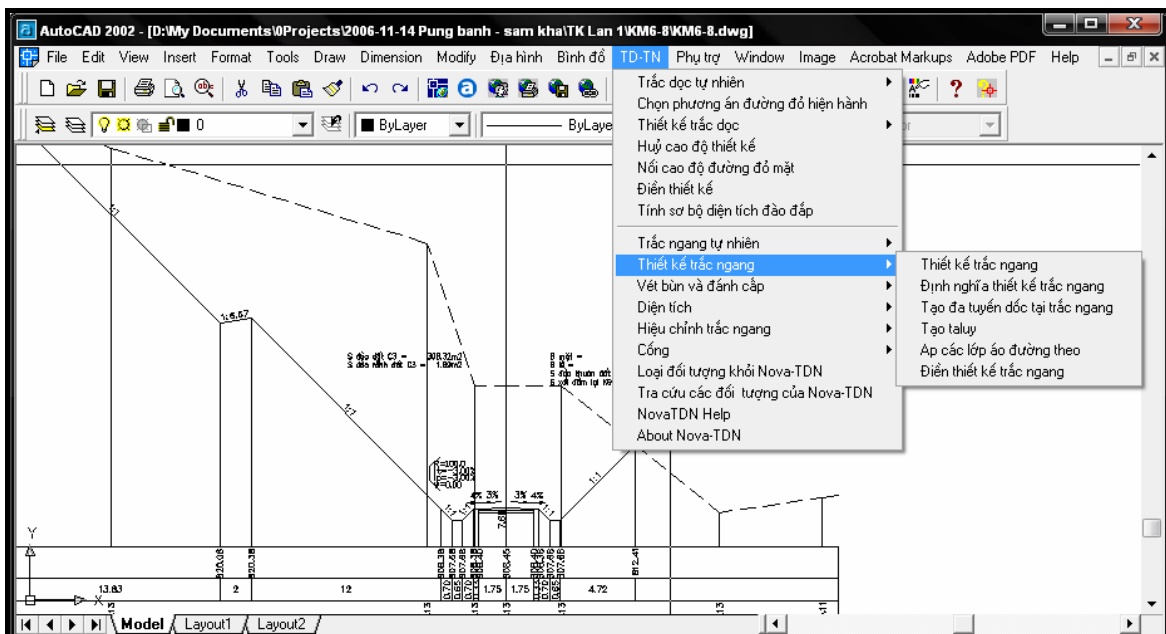
## 4. Chuyên biệt hóa phần mềm

Khi được trang bị phần mềm với mục đích tự động hóa công tác thiết kế thì ta mới giải quyết được các bài toán cơ bản trong quá trình thiết kế, bởi không có phần mềm nào, mà ngay từ đầu, lại có thể đáp ứng được mọi vấn đề sẽ xuất hiện sau này, còn rất nhiều vấn đề mới sẽ liên tục phát sinh trong quá trình thiết kế những công trình cụ thể. Nói cách khác, việc trang bị phần mềm nào đó chỉ là bước đầu cho quá trình tự động hóa, nhưng đây là bước đi quan trọng nhất. Có nhiều cách giải quyết các vấn đề phát sinh này, mà cơ bản và tốt nhất là hai giải pháp:

- ◆ Phản hồi những vấn đề phát sinh cho nhà sản xuất phần mềm để họ nâng cấp phiên bản, sau đó cập nhật lại. Giải pháp này thường mất nhiều thời gian và trong nhiều trường hợp là không khả thi.
- ◆ Tự bổ sung thêm những khả năng mới cho phần mềm đang sử dụng để chúng có thể giải quyết được vấn đề phát sinh. Giải pháp này đòi hỏi phải có nhân lực am hiểu về chuyên môn cầu đường và công nghệ thông tin, đồng thời phần mềm đang sử dụng phải cho phép cập nhật tính năng mới từ phía người dùng. Nhân lực đáp ứng được yêu cầu này chính là kỹ sư xây dựng công trình giao thông được trang bị thêm những kiến thức về tin học phù hợp, đây là mục tiêu chính của môn học Tự động hóa thiết kế cầu đường và cũng là mục tiêu của chính giáo trình này.

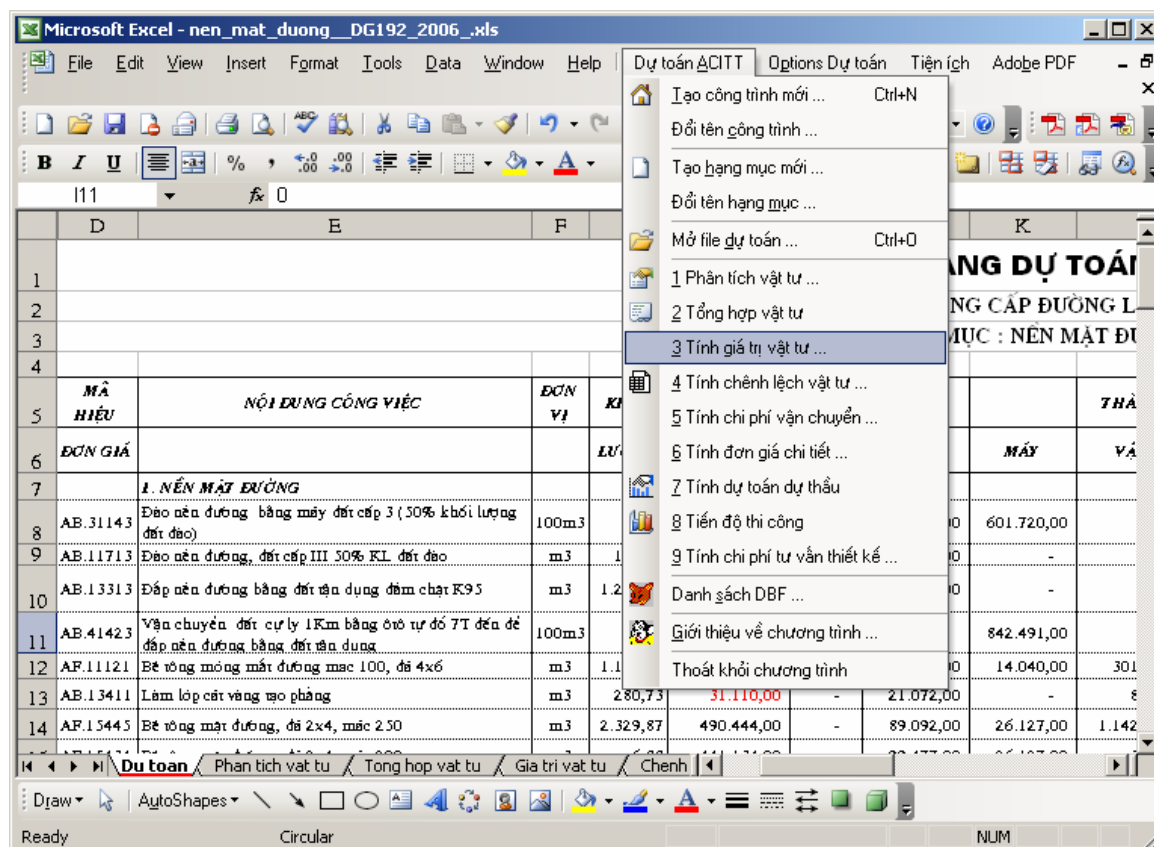
Phần mềm, mà người dùng có thể tự tạo thêm các khả năng mới cho nó, phải có một số đặc điểm sau:

- Cung cấp tính năng cho phép người dùng có thể tự mình bổ sung thêm chức năng cho chính phần mềm đó. Ví dụ phần mềm AutoCAD cho phép người dùng sử dụng công cụ lập trình, như AutoLISP hay ObjectARX, để tự xây dựng thêm những chức năng mới trong AutoCAD.



**Hình I-5: Bổ sung tính năng mới cho AutoCAD**

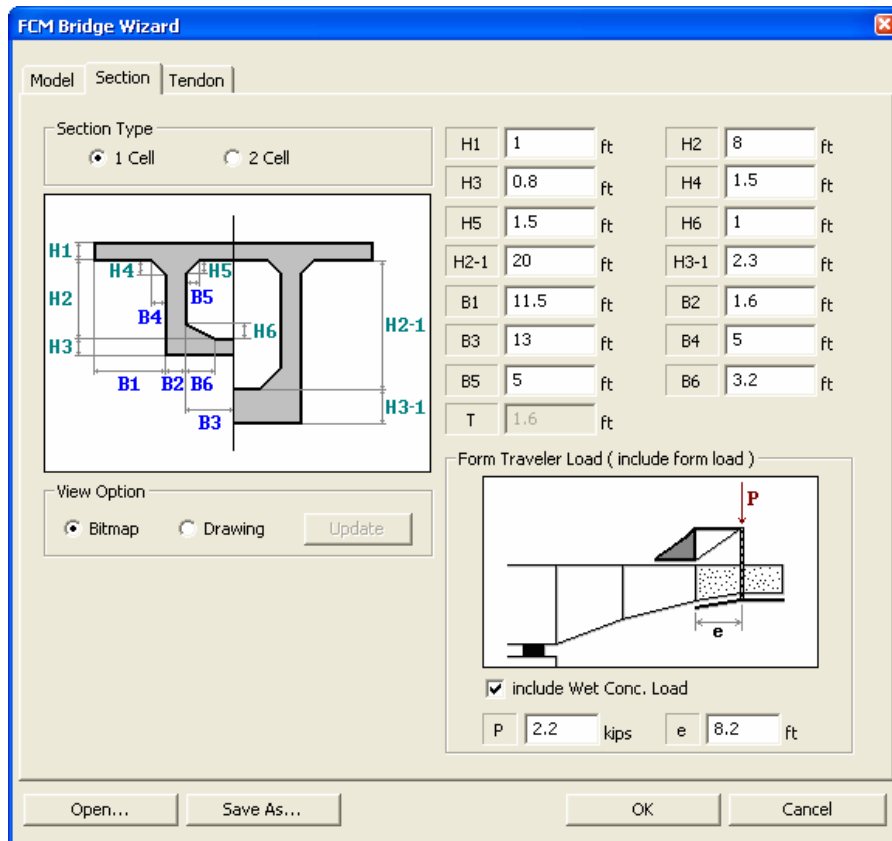
- Cho phép nhúng các phần mềm dạng Add-in vào bên trong, ví dụ như các chương trình trong bộ MS.Office (Excel, Word, Power Point ...). Các chương trình dạng Add-in có thể được xây dựng từ một số công cụ lập trình (ví dụ ta có thể dùng VSTO - Visual Studio Tools for Office - để xây dựng các chương trình dạng Add-in nhúng vào trong bộ Office)



Hình I-6: Bổ sung thêm chức năng lập dự toán cho Excel

- Số liệu đầu vào và kết quả được lưu trữ trên tệp với định dạng có thể hiểu được. Những chương trình dạng này chỉ cho phép người dùng tạo ra những tính năng mới phục vụ cho việc nhập dữ liệu (các chương trình dạng Wizard<sup>1</sup>) hoặc trình bày kết quả.

<sup>1</sup> **Wizard**: thường được hiểu là một chương trình có chức năng trợ giúp người dùng nhập dữ liệu (nhanh và tránh sai sót), nó đặc biệt hữu ích khi dùng những phần mềm đa năng, bởi những phần mềm này thường hay yêu cầu người dùng đưa vào rất nhiều loại dữ liệu mà nhiều khi chúng không thực sự cần thiết cho một bài toán cụ thể. Chương trình dạng Wizard sẽ tự động lọc những thông tin cần thiết cho bài toán cụ thể (để người dùng chỉ cần nhập những dữ liệu cần thiết cho bài toán của mình) còn những số liệu khác mà phần mềm yêu cầu sẽ được chương trình Wizard tự động bổ sung. Bên cạnh đó chương trình Wizard còn có chức năng dẫn dắt người dùng thực hiện bài toán theo một trình tự nhất định để tránh nhầm lẫn.

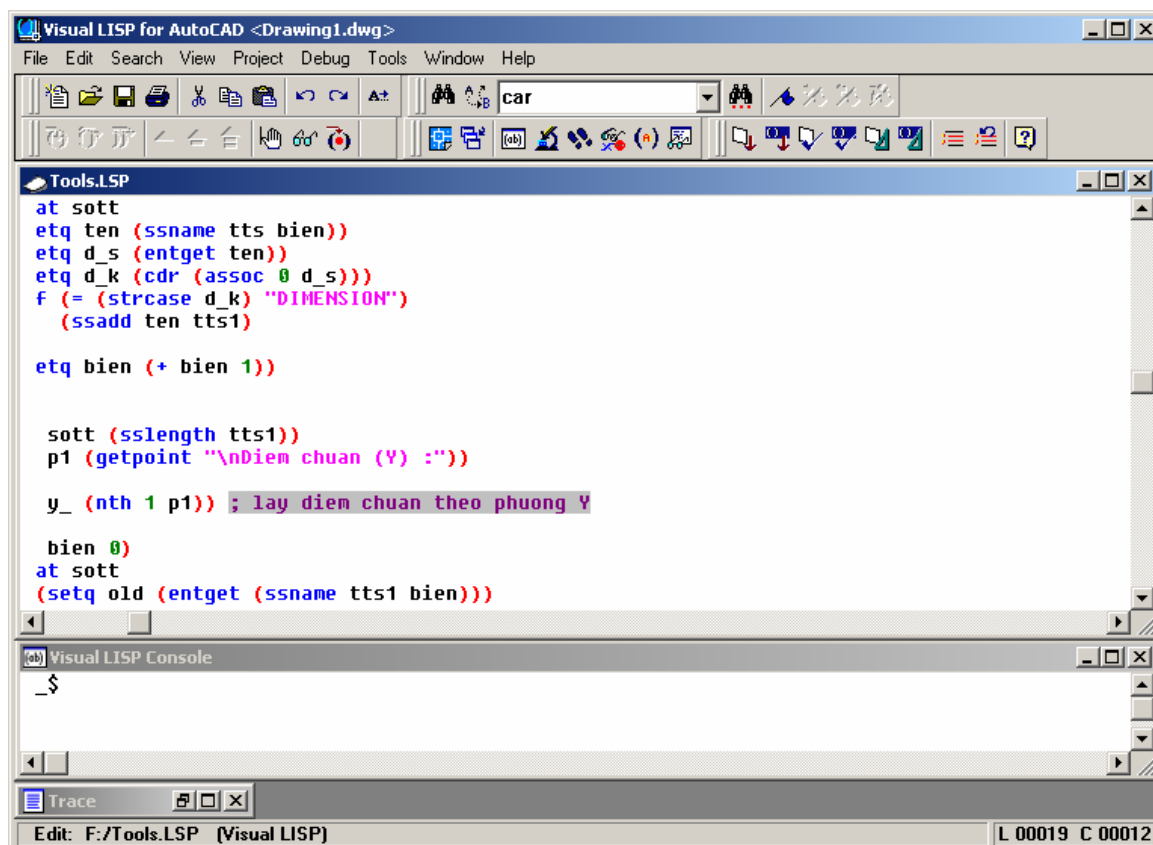


**Hình I-7: Wizard trợ giúp nhập dữ liệu cho kết cấu cầu đúc hằng của MIDAS/Civil**

Công cụ lập trình để tạo ra các tính năng mới cho phần mềm hiện có rất nhiều và khá dễ dùng. Hầu hết chúng tập trung hỗ trợ cho AutoCAD và Office, bởi hai phần mềm này được dùng rất phổ biến trong công tác thiết kế. Với AutoCAD ta có thể sử dụng những công cụ sau:

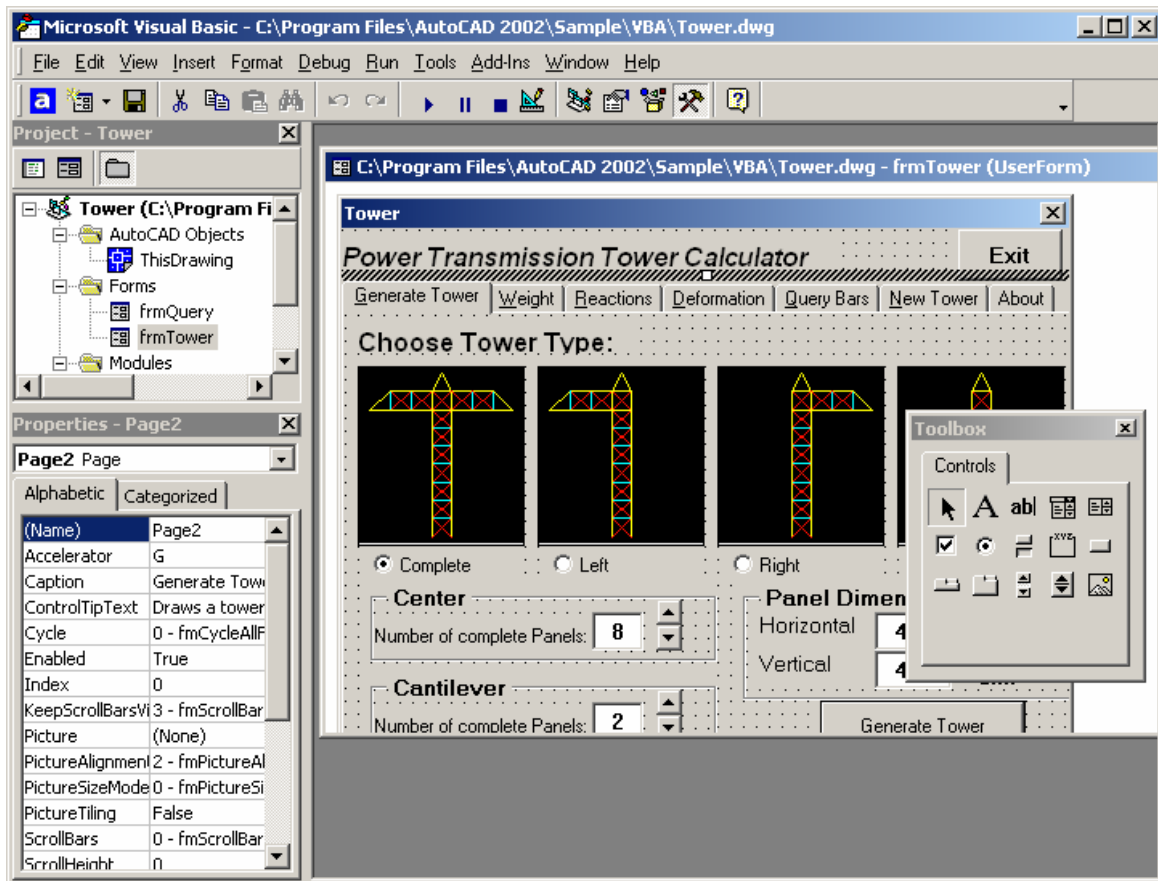
- Các công cụ lập trình nhúng sẵn bên trong AutoCAD:
  - ◆ AutoLISP: là một ngôn ngữ lập trình dạng thông dịch, cho phép người dùng tận dụng tối đa những lệnh sẵn có của AutoCAD để tổ hợp lại nhằm tạo ra những tính năng mới có mức độ tự động hóa cao.





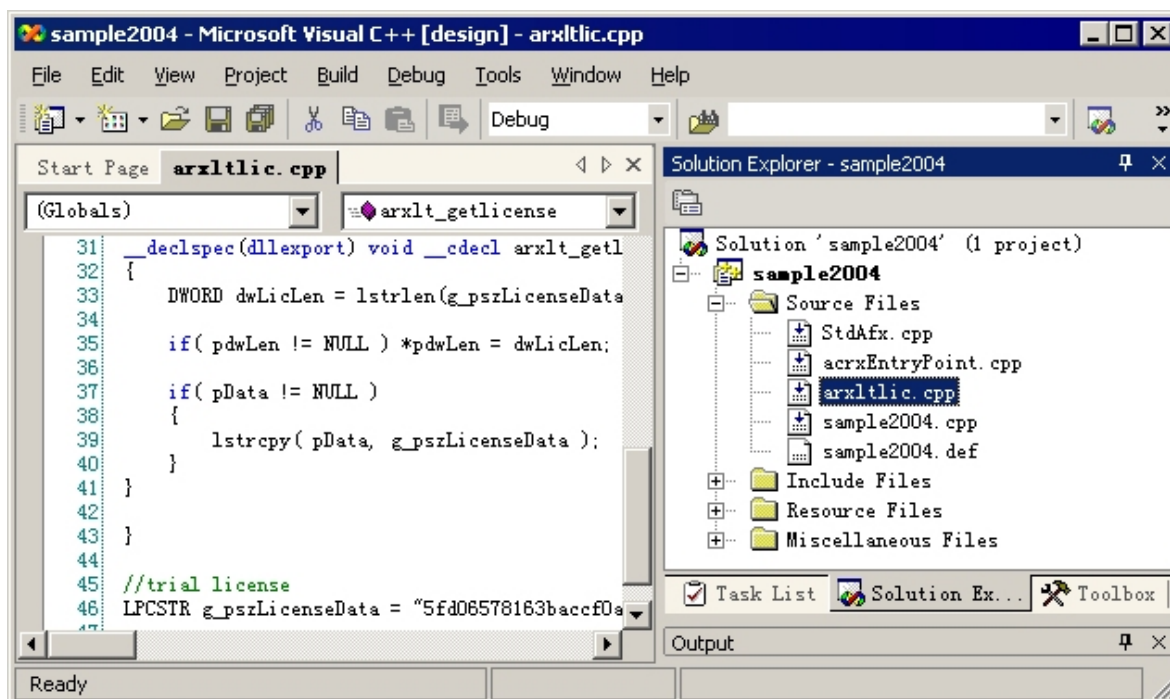
**Hình I-8: Visual LISP: công cụ hỗ trợ cho lập trình với AutoLISP trong AutoCAD**

- ♦ VBA: là một công cụ lập trình dựa trên Visual Basic, nó cho phép người dùng kết hợp tính dễ dùng và hiệu quả của môi trường lập trình Visual Basic với các tính năng và hệ thống đối tượng sẵn có trong AutoCAD. Hiện nay đây là công cụ được dùng rất phổ biến để xây dựng thêm những tính năng mới, với quy mô không lớn và không quá phức tạp trên AutoCAD. Trong lĩnh vực thiết kế công trình giao thông, công việc chiếm khối lượng lớn nhất và mất nhiều công nhất là tạo bản vẽ kỹ thuật. Mặc dù hầu hết người thiết kế đều dùng AutoCAD để tạo bản vẽ kỹ thuật nhưng mức độ tự động hóa vẫn rất thấp, chủ yếu sử dụng các lệnh đơn của AutoCAD (thông qua dòng lệnh hay nút bấm trong AutoCAD) cùng với các thông số hình học tính toán được (có thể bằng các phần mềm khác, ví dụ phần mềm tính kết cấu) để xây dựng bản vẽ. Vấn đề này hoàn toàn có thể tự động hóa được khi người dùng biết kết hợp quy tắc vẽ đối tượng thiết kế với số liệu hình học tính được trong một chương trình VBA do chính họ tạo ra.



**Hình I-9: Môi trường lập trình VBA trong AutoCAD**

- Công cụ lập trình bên ngoài: bao gồm bất cứ ngôn ngữ lập trình nào mà có hỗ trợ công nghệ COM (Component Object Model) của Microsoft như: VB, VC++, Delphi....
- Công cụ lập trình ObjectARX (AutoCAD Runtime Extension): là một cách mở rộng AutoCAD hiệu quả nhất và phức tạp nhất. Các phần mở rộng AutoCAD được xây dựng trên VC++ với việc sử dụng các thư viện lập trình mở rộng của AutoCAD (chính là ObjectARX). Bởi việc cho phép điều khiển trực tiếp nhân và cấu trúc dữ liệu của chương trình AutoCAD, cho nên những chương trình được viết với ObjectARX sẽ có tính linh hoạt rất cao, tốc độ chạy nhanh và nhỏ gọn hơn so với chương trình cùng loại viết bằng công cụ lập trình khác, nhưng mức độ phức tạp của việc lập trình sẽ tăng lên. Hầu hết các ứng dụng lớn chạy trên nền AutoCAD đều được xây dựng dựa trên ObjectARX: Land Desktop, Civil 3D, Nova-TDN...



Hình I-10: Mở rộng khả năng cho AutoCAD dùng ObjectARX

## 5. Kết chương

Như vậy, trong chương này, toàn cảnh về việc ứng dụng công nghệ thông tin để tự động hóa công tác thiết kế công trình giao thông đã được đề cập đến. Vấn đề cốt lõi để tự động hóa thiết kế bao gồm:

- ◆ Quá trình thiết kế công trình giao thông và sản phẩm của từng công đoạn.
- ◆ Khả năng của phần cứng máy tính và các hệ thống phần mềm, bao gồm cả các phần mềm chuyên dụng.
- ◆ Sự đa dạng của các bài toán thiết kế cũng như những hạn chế trong các phần mềm chuyên dụng.
- ◆ Những đặc điểm của phần mềm và các công cụ phát triển, để từ đó có được định hướng trong việc giải quyết các vấn đề phát sinh, vốn thường gặp suốt quá trình thiết kế.

Trong khuôn khổ giáo trình của một môn học, nhiều mảng kiến thức sẽ được kế thừa từ những môn học khác là điều đương nhiên, và do đó, chỉ có những nội dung mới, chưa được đề cập đến trong những môn học khác, mới được trình bày chi tiết ở đây. Với các chương tiếp theo trong giáo trình này, những kiến thức chi tiết để thực hiện tự động hóa thiết kế cầu đường sẽ được đưa ra theo những ý chính của chương đầu tiên này.

## PHẦN II: LẬP TRÌNH TRÊN ỨNG DỤNG NỀN

### CHƯƠNG I: KHÁI NIỆM

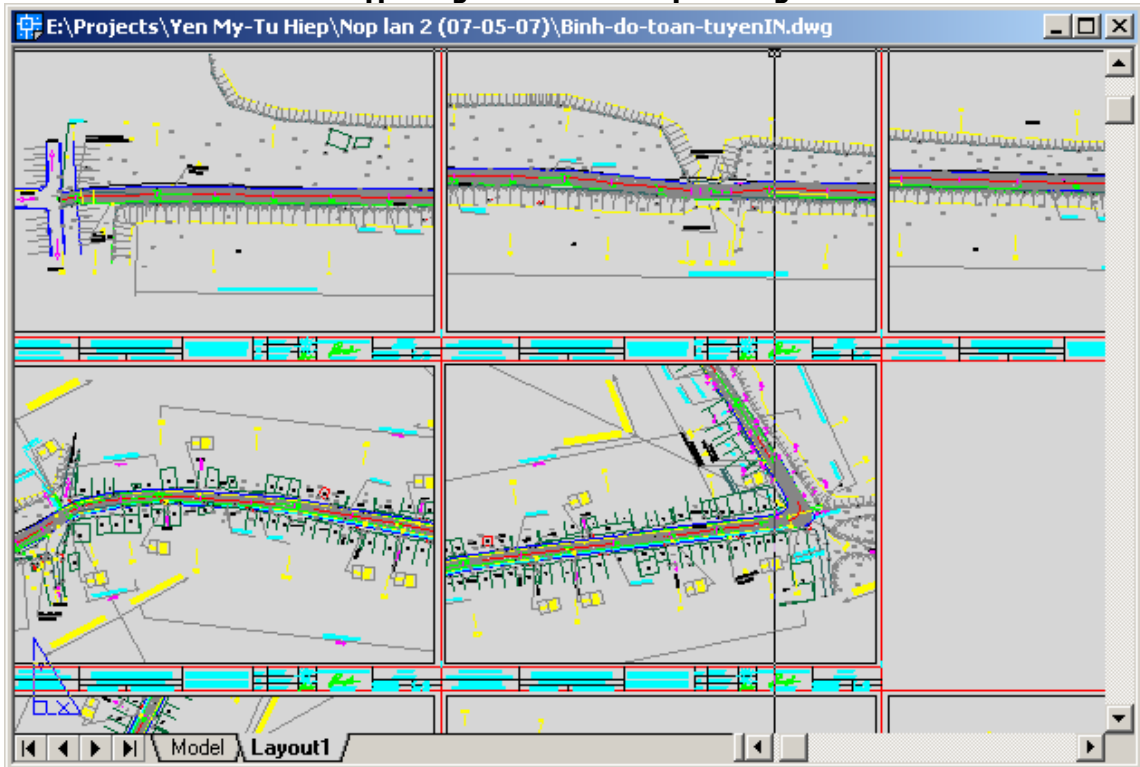
Trong hồ sơ thiết kế, phần tài liệu được trình bày dưới dạng bảng biểu (bảng tính kết cấu, bảng tính khối lượng, ...) và bản vẽ (mô tả cấu tạo hình học của công trình) chiếm một khối lượng đáng kể. Nội dung của những tài liệu trong phần này lại luôn có mối quan hệ rõ ràng và chặt chẽ với phần tính toán trong quá trình thiết kế, chính vì vậy, khả năng thực hiện tự động hóa công đoạn này là hoàn toàn khả thi và mang lại hiệu quả cao. Những công việc cụ thể có thể tự động hóa bao gồm: tính toán, lập bảng tính, lập bản vẽ, trong đó, phần tính toán tạo tiền đề cho quá trình thực hiện lập bảng tính và bản vẽ.

Phần tính toán có thể được tách ra thành một mô-đun riêng và thực hiện độc lập với bất cứ công cụ lập trình nào, và hiện nay, công nghệ lập trình cho phép dễ dàng kết nối các mô-đun loại này với các ứng dụng khác. Phần lập bảng tính và bản vẽ, thực chất sử dụng kết quả thực hiện của mô-đun tính toán và thể hiện kết quả này dưới dạng bản vẽ kỹ thuật và bảng tính, bảng biểu phù hợp với các quy định về trình bày tài liệu trong hồ sơ thiết kế. Trong nhiều trường hợp người ta có thể kết hợp mô-đun tính toán vào cùng với quá trình tạo bảng tính hay bản vẽ, cách làm này rất hiệu quả đối với các bài toán không quá phức tạp về tính toán (như thiết kế hình học đường ô tô hay tính duyệt mặt cắt kết cấu). Nhưng đối với các bài toán có độ phức tạp cao trong tính toán (như bài toán tính kết cấu hay ổn định trượt mái dốc) thì mô-đun tính toán thường được tách riêng ra và kết quả tính toán sẽ được trình bày bởi mô-đun tạo bản vẽ và mô-đun tạo bảng tính riêng. Trong khuôn khổ giáo trình này, do nhắm đến tính phổ biến của các bài toán thông thường có độ phức tạp không cao nhưng đa dạng, cho nên việc định hướng giải quyết bài toán hướng đến việc hợp nhất phần tính toán vào trong mô-đun tạo bảng tính hay mô-đun tạo bản vẽ.

Do bảng tính và bản vẽ có cấu trúc tài liệu rất khác biệt, cho nên hầu như không có phần mềm nào có thể hỗ trợ tốt cho cả hai mục đích trên cùng lúc, và trong thực tế, người ta sử dụng những phần mềm riêng để tạo bản vẽ hay bảng tính. Ví dụ trong lĩnh vực thiết kế công trình giao thông, Excel thường được dùng như là phần mềm hỗ trợ tạo bảng tính chuyên nghiệp, trong khi đó, AutoCAD lại thường được sử dụng trong việc tạo bản vẽ kỹ thuật. Bên cạnh AutoCAD và Excel, còn có nhiều phần mềm chuyên dụng khác, mà khả năng của chúng tập trung vào một số lĩnh vực hẹp, ví dụ như MIDAS/Civil tập trung vào lĩnh vực phân tích kết cấu, Nova-TDN tập trung vào lĩnh vực thiết kế hình học đường ô tô. Kết quả mà các phần mềm chuyên dụng này mang lại khá đầy đủ, có thể bao gồm hầu hết các bảng tính và bản vẽ liên quan đến bài toán được giải quyết. Tuy vậy, trong phạm vi lĩnh vực của mình, không phần mềm chuyên dụng nào có thể đáp ứng được mọi nhu cầu, và do đó, chúng thường được thiết kế theo hướng có thể kết nối với các phần mềm khác nhằm mục đích hỗ trợ người dùng giải quyết được vấn đề phát sinh bằng cách kết hợp vài phần mềm với nhau.

	A	B	C	D	E	F	G	H
1	<b>II. PHÂN TÍNH TOÁN KẾT CẤU:</b>							
4	Chiều dày lớp mặt dự kiến h(cm)=		25	BTXM M250 ( $\sigma_u$ ) = 35 daN/cm <sup>2</sup> , $E_b = 29 \times 10^4$ daN/cm <sup>2</sup>				
5	Chiều dày lớp móng h <sub>m</sub> (cm)=		15	BTXM M100	$E_m =$	135000	daN/cm <sup>2</sup>	
6	Vậy $D = D_o + h = 36 + 25 =$		61 cm		$E_o =$	400,0	daN/cm <sup>2</sup>	
7								
8	<b>2./ Tính chiều dày tấm bê tông xi măng:</b>							
9		$D = D_o + h =$	61	cm				
10	$h_m$	15			$E_o$	400		
11	$\frac{\quad}{\quad} =$	$\frac{\quad}{\quad} =$	0,25	và	$\frac{\quad}{\quad} =$		0,003	
12	D	61			$E_{ch}$	135.000		
13	Tra toán đồ H3-3 ta có :							
14	$E_{ch}^m$				$E_{ch}^m$			
15	$\frac{\quad}{\quad} =$	0,030	Suy ra	$E_{ch}^m = E_{ch} \times$	$\frac{\quad}{\quad} =$	4050	daN/cm <sup>2</sup>	
16	$E_{ch}$				$E_{ch}$			
17								
18	Xác định các hệ số $\alpha_1, \alpha_2, \alpha_3$ theo vị trí đặt tải :							
19								
20				2R				
21								
22								
23		2R						
24			2R					
25								
26								
27								

Hình I-1: Lập bảng tính kết cấu mặt đường trên Excel



Hình I-2: Tạo bản vẽ bình đồ tuyến đường ô tô trên AutoCAD

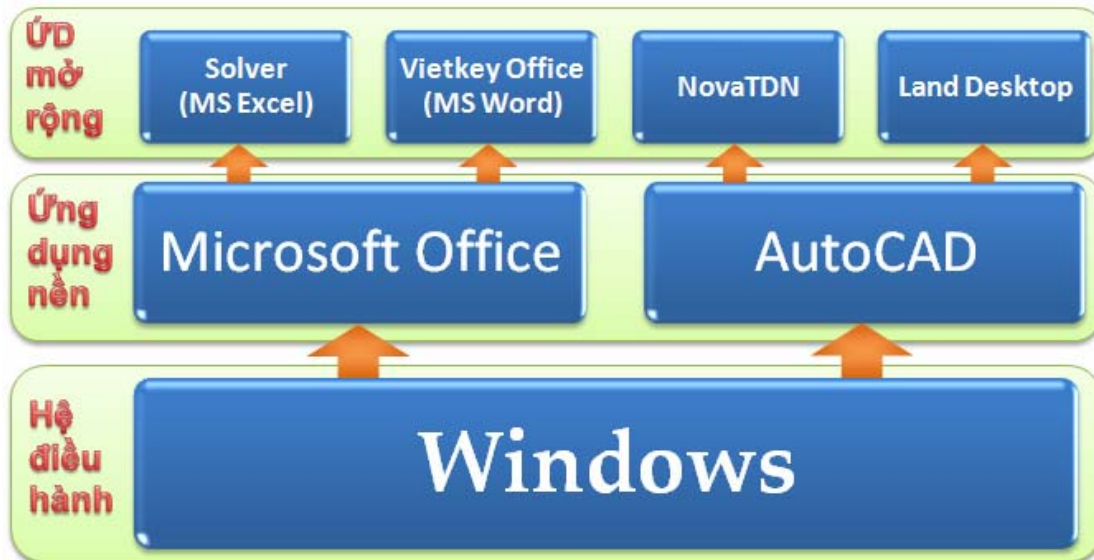
Để có thể kết nối với nhau, các phần mềm chuyên dụng thường cung cấp kết quả tính toán dưới dạng dữ liệu có cấu trúc và được lưu trữ trong các tệp có định dạng TEXT, ví dụ như CSV hay DXF. Với các dữ liệu có cấu trúc này, người dùng sẽ tự thực hiện việc kết nối các phần mềm lại với nhau. Việc kết nối này cũng chỉ có thể giải quyết thêm một số bài toán phát sinh, cho nên một số phần mềm đã cho phép người dùng có thể can thiệp sâu hơn nữa vào bên trong nó bằng các công cụ lập trình, để họ có thể tự giải quyết các bài toán phát sinh mà người thiết kế phần mềm không thể dự kiến trước được. Khi người dùng xây dựng những chương trình của họ dựa trên những ứng dụng được thiết kế theo cấu trúc mở này, họ sẽ tận dụng những khả năng

## GIÁO TRÌNH TỰ ĐỘNG HOÁ THIẾT KẾ CẦU ĐƯỜNG

sẵn có của chúng để làm nền, giúp cho việc lập trình được nhanh và hiệu quả hơn rất nhiều so với cách lập trình thông thường, và do đó, có thể gọi chúng là các **ứng dụng nền**, điển hình và được sử dụng nhiều nhất làm ứng dụng nền trong lĩnh vực thiết kế là AutoCAD và Excel, ngoài việc phù hợp với định dạng tài liệu trong hồ sơ thiết kế (bản vẽ và bảng tính) chúng còn cho phép người dùng xây dựng các chương trình chạy cùng với mục đích bổ sung thêm các chức năng chuyên biệt.

Như vậy, một phần mềm được gọi là **ứng dụng nền** khi nó thỏa mãn đồng thời các tiêu chí sau:

- ◆ Cho phép một chương trình chạy bên trong và cùng với nó (tương tự như một lệnh).
- ◆ Cho phép sử dụng các tính năng của nó thông qua công cụ lập trình thích hợp.



**Hình I-3: Mô hình lập trình trên ứng dụng nền**

Một lệnh mới hay một chức năng mới được xây dựng trên ứng dụng nền thực chất là một chương trình hoàn chỉnh, vì vậy, để xây dựng nó cần có công cụ lập trình tương ứng. Thông thường **công cụ lập trình** được hiểu như là một tập hợp bao gồm:

- ◆ Ngôn ngữ lập trình.
- ◆ Môi trường lập trình.
- ◆ Thư viện hỗ trợ lập trình.

Một ví dụ về công cụ lập trình trên AutoCAD, đó là AutoLISP. Với công cụ lập trình này, không nhất thiết phải có môi trường lập trình và thư viện hỗ trợ lập trình, ta chỉ cần tạo ra một tệp dạng TEXT chứa các mã lệnh viết bằng ngôn ngữ AutoLISP. Tuy nhiên từ phiên bản AutoCAD R14, để thuận tiện cho người lập trình, một môi trường lập trình dành cho AutoLISP đã được bổ sung, đó là Visual LISP. Với môi trường lập trình này, việc lập và kiểm soát chương trình trở nên thuận lợi hơn rất nhiều, bởi Visual LISP đã được tích hợp nhiều tính năng hỗ trợ lập trình chuyên nghiệp, trong khi đó, nếu ta không sử dụng môi trường lập trình, thì tuy ta có thể viết được một chương trình AutoLISP hoàn chỉnh, song trong suốt quá trình xây dựng chương trình này ta luôn phải vất vả để tự kiểm soát chương trình.

```

Lister - [C:\DOCUME~1\LEQUYN~1\LOCALS~1\Temp\_t...
File Edit Options Help 22 %
(defun C:TH ()
  (prompt "\n Chon DIM can thay doi :\n")
  (setq tts (ssget))
  (setq sott (sslenght tts))

  ;filter objects
  (setq tts1 (ssadd))
  (setq bien 0)
  (repeat sott
    (setq ten (ssname tts bien))
    (setq d_s (entget ten))
    (setq d_k (cdr (assoc 0 d_s)))
    (if (= (strcase d_k) "DIMENSION")
      (ssadd ten tts1)
    )
    (setq bien (+ bien 1))
  )
)
    
```

**Hình I-4:** Xây dựng chương trình bằng ngôn ngữ AutoLISP khi không sử dụng môi trường lập trình, ta sẽ luôn phải tự kiểm soát cú pháp và các lệnh mà không có bất cứ hỗ trợ nào vì thế khả năng nhầm lẫn là rất lớn.

Visual LISP for AutoCAD <Drawing1.dwg>

File Edit Search View Project Debug Tools Window Help

car

TOOL.LSP

```

(defun C:TH ()
  (prompt "\n Chon DIM can thay doi :\n")
  (setq tts (ssget))
  (setq sott (sslenght tts))

  ;filter objects
  (setq tts1 (ssadd))
  (setq bien 0)
  (repeat sott
    (setq ten (ssname tts bien))
    (setq d_s (entget ten))
    (setq d_k (cdr (assoc 0 d_s)))
    (if (= (strcase d_k) "DIMENSION")
      (ssadd ten tts1)
    )
    (setq bien (+ bien 1))
  )
)
    
```

Visual LISP Console

\_\$

Edit: E:/Le Quynh Mai/Giao trinh/TOOL.LSP [Visual LISP] L 00054 C 00008

**Hình I-5:** Lập trình bằng ngôn ngữ AutoLISP trên môi trường lập trình Visual LISP, ta luôn nhận được sự hỗ trợ tự động bằng màu sắc hay các tính năng khác trong môi trường lập trình.

Thư viện hỗ trợ lập trình có thể rất đa dạng và thường là những phần bổ sung giúp cho việc xây dựng chương trình được nhanh hơn thông qua sự kế thừa những thứ đã được làm từ trước. Khi lập trình bằng AutoLISP thì thư viện hỗ trợ lập trình là tập hợp các chương trình hoàn chỉnh cũng viết bằng AutoLISP. Để sử dụng thư viện hỗ trợ lập trình thì mỗi công cụ lập trình có một quy định về cách thức sử dụng riêng, ví dụ với AutoLISP, để sử dụng một chương trình con trong thư viện, ta chỉ cần tải chương trình AutoLISP chứa chương trình con đó thông qua một câu lệnh từ chương trình chính.

Tương ứng với từng ứng dụng nền thì sẽ có các công cụ lập trình phù hợp. Một ứng dụng nền có thể hỗ trợ một hay nhiều công cụ lập trình khác nhau, tùy mục đích sử dụng. AutoCAD hỗ trợ các công cụ lập trình trên ứng dụng nền sau:

- ◆ AutoLISP
- ◆ ObjectARX
- ◆ VBA (Visual Basic for Applications)

Còn Excel hỗ trợ các công cụ lập trình:

- ◆ VBA
- ◆ VSTO (Visual Studio Tools for Office)

Mỗi công cụ lập trình luôn có những đặc điểm riêng và khó có thể phán xét cái nào hay hơn hoặc kém hơn một cách tổng quát. Do đó, để lựa chọn được công cụ lập trình thích hợp khi lập trình trên ứng dụng nền, cần dựa vào mục đích cụ thể. Ví dụ khi lập trình trên AutoCAD, để tạo các công cụ trợ giúp vẽ thì AutoLISP là lựa chọn hợp lý. Nhưng để xây dựng những ứng dụng lớn, phức tạp, đòi hỏi phải can thiệp sâu vào bên trong AutoCAD thì chỉ có thể dùng ObjectARX mới làm được.

Trong lĩnh vực tự động hóa thiết kế công trình giao thông, hầu hết các bài toán lớn và cơ bản đã được giải quyết, nhưng còn rất nhiều các bài toán khác, tuy không lớn và không quá phức tạp, nhưng lại rất đa dạng và khó khái quát, vẫn chưa có phần mềm thực hiện, và do đó, phạm vi ứng dụng của lập trình trên ứng dụng nền là rất lớn và có tính hiệu quả cao. Hơn nữa, với quy mô của các bài toán này, thì việc lựa chọn VBA làm công cụ lập trình là rất phù hợp bởi:

- ◆ Ngôn ngữ lập trình Visual Basic (VB) là một loại ngôn ngữ dễ sử dụng, có số lượng người dùng đông đảo và tài liệu tham khảo rất phong phú. Điều này cho phép người dùng trao đổi kỹ năng, tìm kiếm tài liệu, mã nguồn một cách dễ dàng.
- ◆ Môi trường lập trình thân thiện, dễ dùng và đầy đủ nên việc xây dựng ứng dụng sẽ nhanh và không cần thêm công cụ lập trình nào khác.
- ◆ Trên tất cả các ứng dụng nền hỗ trợ VBA, giao diện lập trình là đồng nhất, 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 nền (chẳng hạn như tệp bảng tính của Excel hay tệp bản vẽ của AutoCAD) 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.

### Kết chương

Tự động hóa công tác lập hồ sơ thiết kế công trình giao thông là hoàn toàn khả thi và có thể được thực hiện theo nhiều cách khác nhau.

Dự án VBA nên xây dựng theo hướng gộp cả phần tính toán và xuất kết quả vào một mô-đun thống nhất.

Sử dụng AutoCAD và Excel làm ứng dụng nền để xây dựng các ứng dụng bằng VBA nhằm mục đích hỗ trợ thiết kế là lựa chọn mang tính khả thi cao và có nhiều ưu điểm.

Để lập trình với VBA, cách tốt nhất, là làm chủ từng phần. Đầu tiên cần nắm vững ngôn ngữ lập trình Visual Basic và cách sử dụng VBA IDE để viết mã lệnh cũng như thiết kế giao diện.



Sau đó nghiên cứu mô hình đối tượng của ứng dụng nền (là những thành phần của ứng dụng nền mà người dùng có thể sử dụng) cũng như cách sử dụng chúng bằng VBA.



## CHƯƠNG II: TỔNG QUAN VỀ VBA

### 1. Đặc điểm của VBA

Từ các đặc điểm cơ bản đã được phân tích ở cuối chương 1 ta 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<sup>1</sup>, 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 ta 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 AutoCAD và VBA trong Excel) là cách thức sử dụng các thành phần (đối tượng) của ứng dụng nền. Cho nên khi xây dựng ứng dụng bằng VBA, việc đầu tiên là phải tìm hiểu mô hình đối tượng của ứng dụng nền và cách sử dụng chúng.

Như trong chương trước đã trình bày, xây dựng một dự án VBA, một cách tổng quát, người dùng cần nắm vững hai phần:

- ◆ Ngôn ngữ lập trình Visual Basic và giao diện lập trình VBA IDE. Phần này sẽ bao gồm các nội dung kiến thức trong chương 2 và 3.
- ◆ Mô hình đối tượng của ứng dụng nền và cách sử dụng chúng. Nội dung kiến thức của phần này sẽ được trình bày trong chương 4 và 5.

### 2. Trình tự xây dựng một dự án bằng VBA

Về mặt trình tự thực hiện, việc xây dựng một dự án VBA bao gồm các bước sau:

1. Xác định rõ nhu cầu xây dựng chương trình. Nhu cầu này được xác định dựa trên hoạt động thực tế của người dùng và thường do chính người dùng đề xuất. Đây là bước xác định các chức năng của chương trình.
2. Xác định rõ mục tiêu mà chương trình cần đạt được. Bước này là phần cụ thể hóa của bước 1, ví dụ như bước 1 có nhu cầu hoàn thiện bản vẽ kết cấu BTCT, còn bước này sẽ cụ thể mức độ hoàn thiện (đến đâu và như thế nào).
3. Lựa chọn ứng dụng nền và công cụ lập trình phù hợp cho việc xây dựng chương trình. Ví dụ với nhu cầu tính và tạo bản vẽ của cấu kiện BTCT, thì ứng dụng nền thích hợp là AutoCAD và công cụ lập trình có thể là AutoLISP, VBA, ObjectARX. Tùy theo mức độ phức tạp của bài toán mà ta lựa chọn công cụ lập trình phù hợp. Ở đây VBA đảm bảo sự thuận tiện trong việc xây dựng các mô-đun tính toán và tạo bản vẽ đối với những bài toán thông thường.
4. Thiết kế hệ thống cho chương trình (hay dự án): bao gồm việc lập sơ đồ khối, xác định các mô-đun của chương trình, thiết kế giao diện nhập xuất dữ liệu và kết quả, xây dựng hệ thống cơ sở dữ liệu sao cho thỏa mãn những đề xuất ở bước 1 và 2.

<sup>1</sup> COM (Component Object Model): là một kiến trúc lập trình được thiết kế bởi Microsoft. Mục đích của công nghệ này là 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.

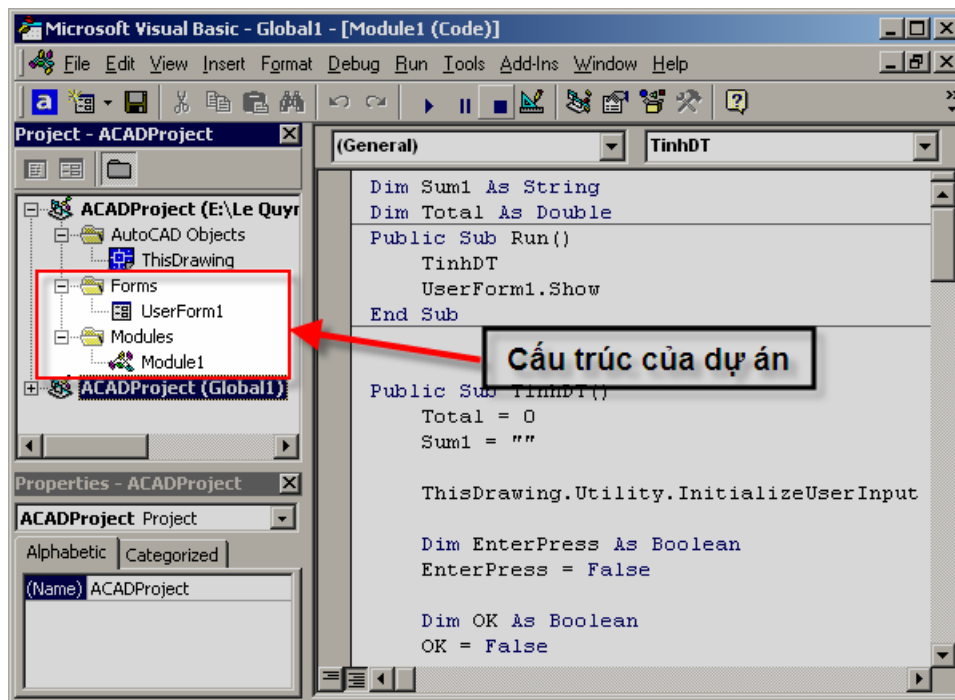
5. Viết mã lệnh (lập trình): là việc sử dụng công cụ lập trình để tạo ra chương trình phù hợp với hệ thống đã được thiết kế ở bước 4.
6. Kiểm thử chương trình: là công đoạn hoàn thiện và chuẩn bị đưa chương trình vào sử dụng. Những công việc chính của bước này bao gồm:
  - ◆ Kiểm tra xem các chức năng của chương trình đã thỏa mãn các yêu cầu đề ra từ trước chưa bằng cách chạy thử tất cả các tính năng của chương trình dựa trên một kịch bản cụ thể.
  - ◆ Kiểm tra hiệu năng của chương trình: xem thời gian thực hiện và quy trình sử dụng chương trình có hợp lý không.
  - ◆ Kiểm tra khả năng chịu lỗi của chương trình, ví dụ như khi nhập số liệu sai. Một chương trình đảm bảo khả năng chịu lỗi là nó sẽ không bị dừng lại đột ngột do lỗi thao tác của người dùng hay dữ liệu sai.
7. Đóng gói, đưa chương trình vào sử dụng: bao gồm việc xây dựng tài liệu hướng dẫn cài đặt và sử dụng chương trình nhằm mục đích giúp người dùng có thể triển khai chương trình vào thực tế.
8. Tiếp nhận các góp ý, phản hồi của người dùng để bổ sung hay hoàn thiện những khiếm khuyết của chương trình mà trong quá trình thiết kế hệ thống hay kiểm thử đã bỏ qua hoặc chưa phát hiện được.
9. Nâng cấp chương trình: sau một thời gian sử dụng, dựa trên những phản hồi của người dùng, nếu thấy rằng chương trình cần bổ sung thêm những tính năng mới thì người phát triển phần mềm sẽ thực hiện sự bổ sung này dựa trên những thành phần đã có từ trước.

### 3. Cấu trúc của 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 thủ tục). 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 dạng hộp thoại 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, thì có thể không cần sử dụng Userform.

Những thành phần này 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.



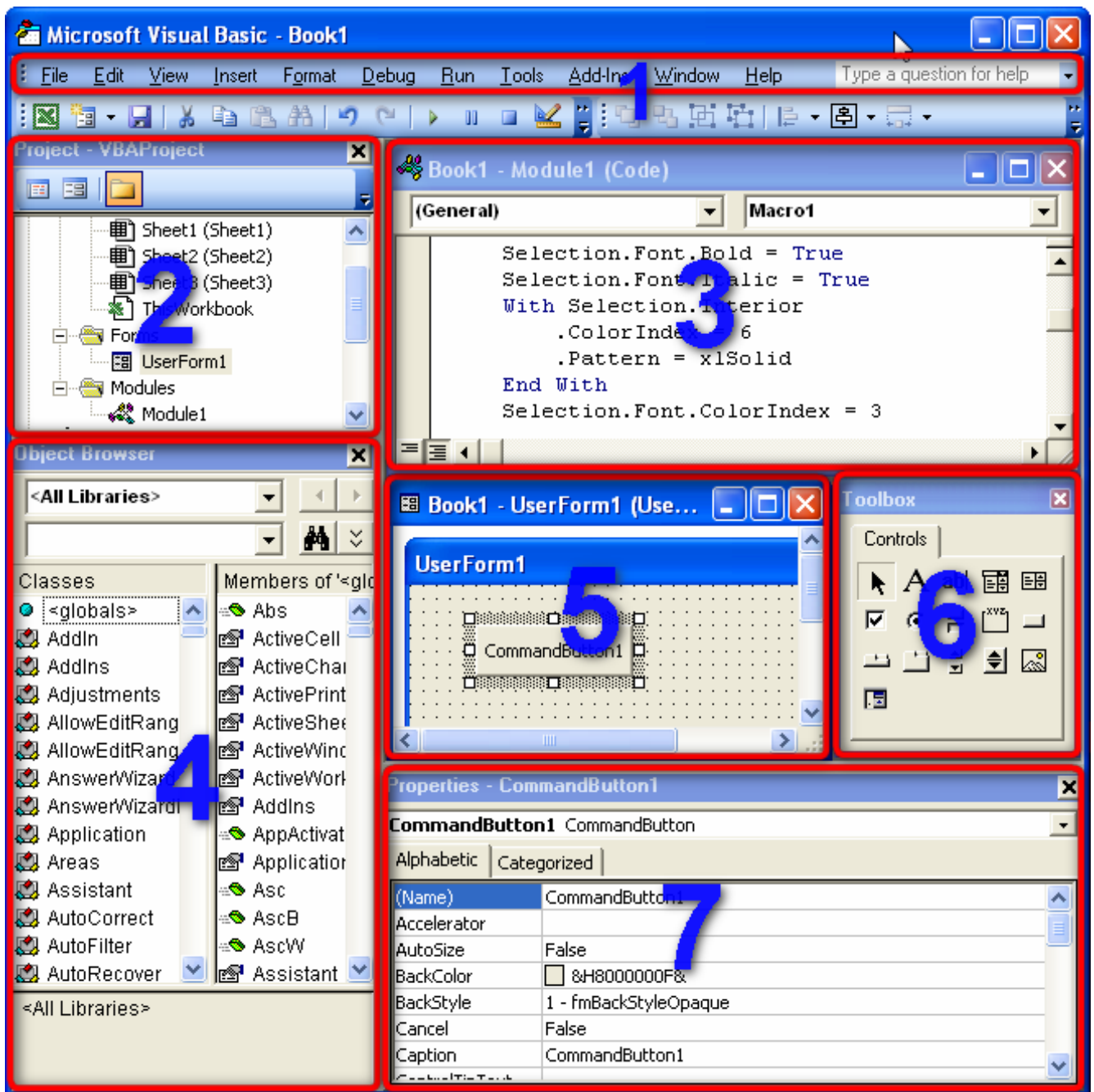
Hình II-1: Cấu trúc của dự án thể hiện trên VBA IDE

Tuy nhiên, khi xây dựng chương trình (viết mã lệnh) cụ thể thì khái niệm cấu trúc của một chương trình là sự bố trí, sắp xếp các câu lệnh trong chương trình đó. Như vậy khái niệm cấu trúc này phụ thuộc vào từng loại ngôn ngữ lập trình. Đối với ngôn ngữ lập trình Visual Basic (VB), cấu trúc của nó chỉ tập trung vào chương trình con (hàm và thủ tục) chứ không có một quy định về cấu trúc nào đối với chương trình chính. Chi tiết của cấu trúc của chương trình con sẽ được đề cập đến trong các phần sau.

#### 4. Môi trường phát triển tích hợp VBA IDE

Trong mỗi công cụ lập trình trên ứng dụng nền, 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 AutoCAD 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). Trên tất cả các ứng dụng nền, VBA IDE có cấu trúc và hoạt động tương đương nhau với giao diện cơ bản và cách gọi giao diện VBA IDE từ ứng dụng nền như sau:

- ◆ Phím tắt: từ giao diện chính của ứng dụng nền, nhấn tổ hợp phím **Alt+F11**.
- ◆ Menu: Tools ⇒ Macro ⇒ Visual Basic Editor.



Hình II-2: Giao diện chính của VBA IDE

1. Thanh trình đơn (Menu bar): chứa tất cả các lựa chọn cần thiết để thao tác với VBA IDE
2. Cửa sổ dự án (Project Explorer Window): liệt kê dưới dạng cây phân cấp các dự án hiện đang được mở trong VBA IDE và các thành phần có trong từng dự án như các tài liệu thành phần, các mô-đun chứa chương trình con, các mô-đun lớp, các cửa sổ do người dùng tạo.



**GỢI Ý** Việc thêm các thành phần mới vào trong một dự án được thực hiện trong menu Insert của VBA IDE. Ví dụ muốn thêm một mô-đun chuẩn vào trong dự án, chọn Insert ⇒ Module

3. Cửa sổ mã lệnh (Code Window): mỗi thành phần được liệt kê trong cửa sổ dự án đều có một cửa sổ mã lệnh riêng, chứa mã lệnh cho thành phần đó. Người dùng có thể hiệu chỉnh mã lệnh, tạo ra mã lệnh mới trong cửa sổ mã lệnh.

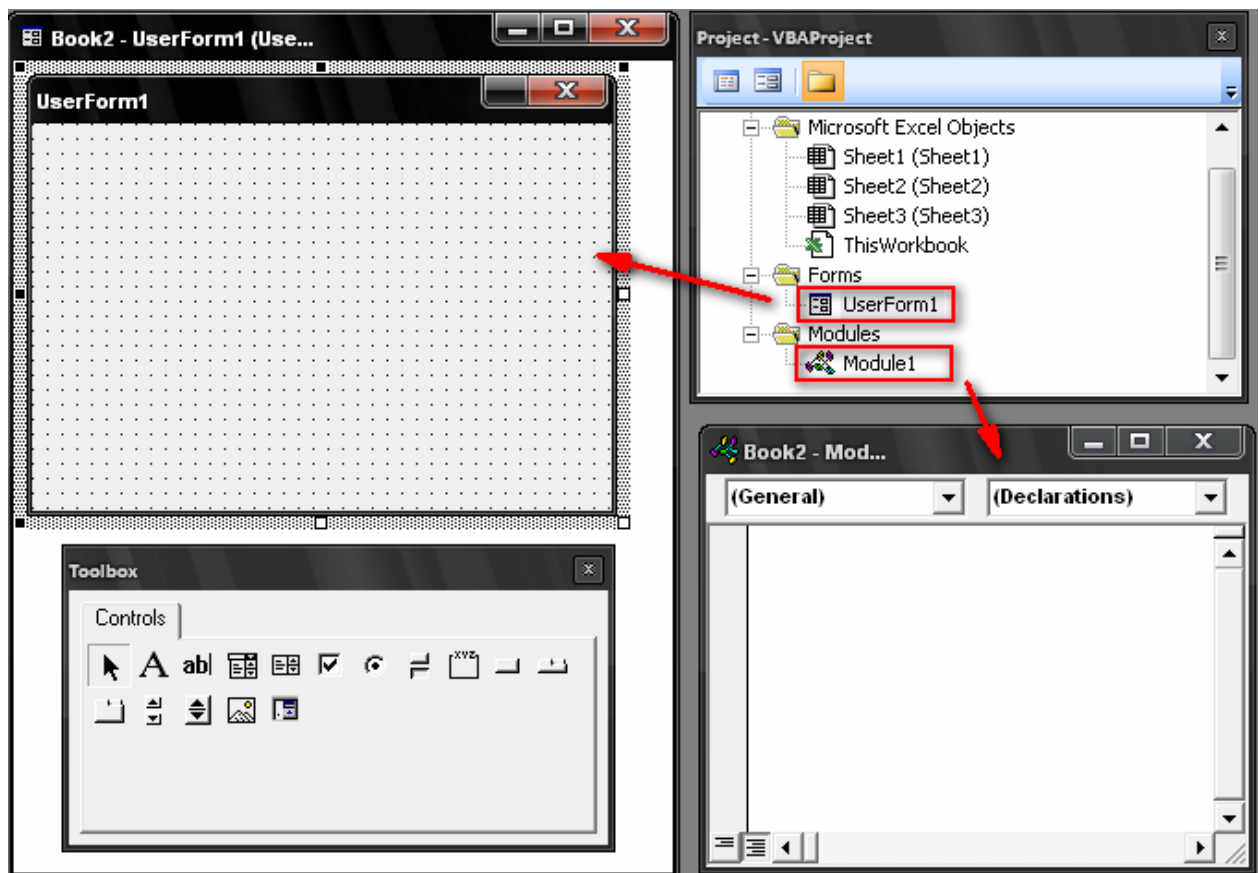
4. Cửa sổ tra cứu đối tượng (Object Browser Window): hiển thị các lớp, phương thức, thuộc tính, sự kiện và hằng số có trong thư viện đối tượng và trong dự án mà người dùng vừa tạo. Ta có thể sử dụng cửa sổ này để tìm kiếm, tra cứu tất cả các đối tượng mà ta vừa tạo ra cũng như các đối tượng trong các chương trình khác.
5. Cửa sổ đối tượng trực quan (Visual Object Window): khi người dùng tạo các đối tượng trực quan thì cửa sổ này sẽ cho phép người dùng thao tác trên các điều khiển một cách dễ dàng và thuận tiện.
6. Hộp công cụ chứa điều khiển (Tool Box): chứa các thanh công cụ giúp người dùng có thể chèn các điều khiển vào cửa sổ người dùng (UserForm).
7. Cửa sổ thuộc tính (Properties Window): cửa sổ này liệt kê tất cả các thuộc tính của đối tượng, qua đó người dùng có thể tham khảo và thay đổi các thuộc tính khi cần như màu chữ, tên đối tượng...

## 5. Ví dụ đầu tiên với VBA

Ví dụ này được trình bày với mục đích giúp người dùng làm quen với VBA IDE trong Excel. Kết quả của ví dụ là hiển thị nội dung ô A1 trong Sheet1 của bảng tính lên tiêu đề của một hộp thoại người dùng (UserForm).

Trình tự thực hiện như sau:

1. Mở ứng dụng Excel, nhấn tổ hợp phím ALT+F11 để vào VBA IDE.
2. Trong VBA IDE, chọn menu Insert ⇒ UserForm để thêm một hộp thoại người dùng vào trong dự án.
3. Chọn tiếp menu Insert ⇒ Module để thêm một mô-đun chuẩn vào trong dự án.

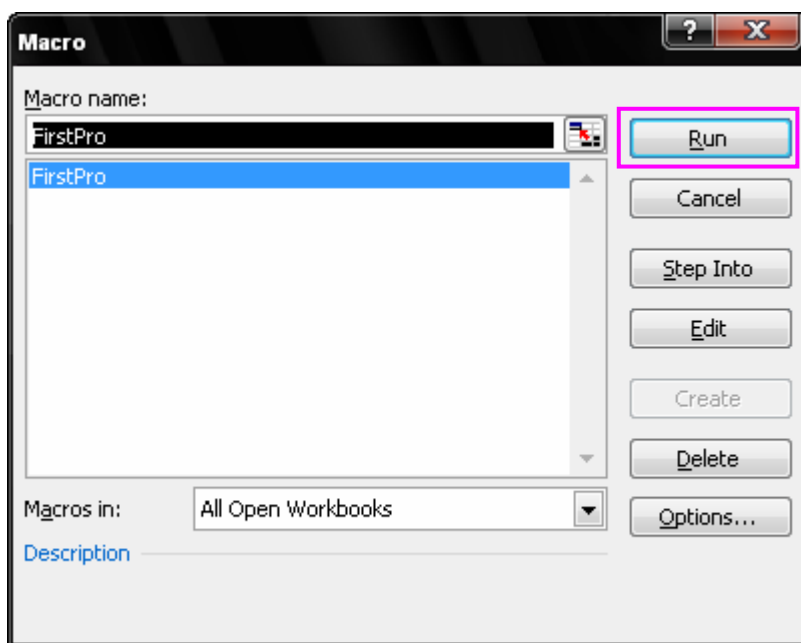


4. Chọn Module1 và soạn thảo mã lệnh trong mô-đun đó như sau:

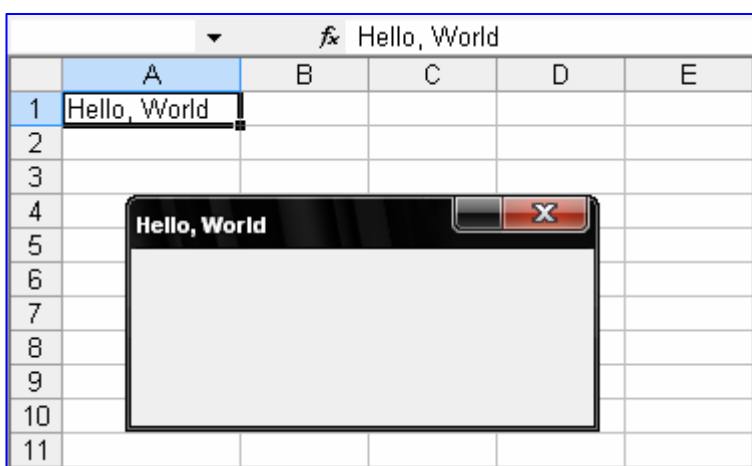
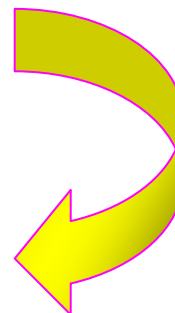
```
Public Sub FirstPro()  
    UserForm1.Show  
    UserForm1.Caption = Sheets("Sheet1").Range("A1").Value  
End Sub
```

Sau đó quay trở lại Excel, và chạy chương trình theo trình tự:

1. Gõ vào ô A1 của Sheet1 nội dung “Hello, World”.
2. Chọn menu Tools ⇒ Macro ⇒ Macros (hoặc nhấn tổ hợp phím ALT+ F8).
3. Trong hộp thoại Macro, chọn macro có tên FirstPro rồi nhấn nút Run. Kết quả chương trình sẽ hiển thị như hình dưới đây:



Hộp thoại Macro



Kết quả trên Excel



## CHƯƠNG III: CƠ BẢN VỀ NGÔN NGỮ LẬP TRÌNH VISUAL BASIC

Trong chương này sẽ trình bày những kiến thức cơ bản trong ngôn ngữ lập trình Visual Basic (VB) như: cú pháp, các từ khoá, các kiểu dữ liệu, các khai báo,... Tất cả các ví dụ sẽ được viết và trình bày kết quả trong VBA IDE.

### 1. Những quy định về cú pháp

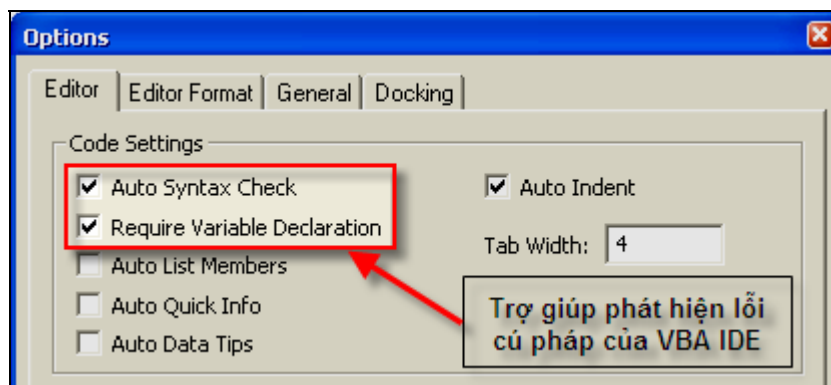
Cú pháp được hiểu là một tập hợp bao gồm các quy tắc, luật lệ về trật tự và hình thức viết của một câu lệnh hay một cấu trúc lệnh.

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 dụng dấu cách và dấu gạch dưới ( \_).
- ✦ Nếu muốn chèn thêm ghi chú, phải bắt đầu dòng chú thích bằng dấu nháy đơn (').
- ✦ Quy 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; không quá 255 kí tự; không trùng với các từ khoá; các biến 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

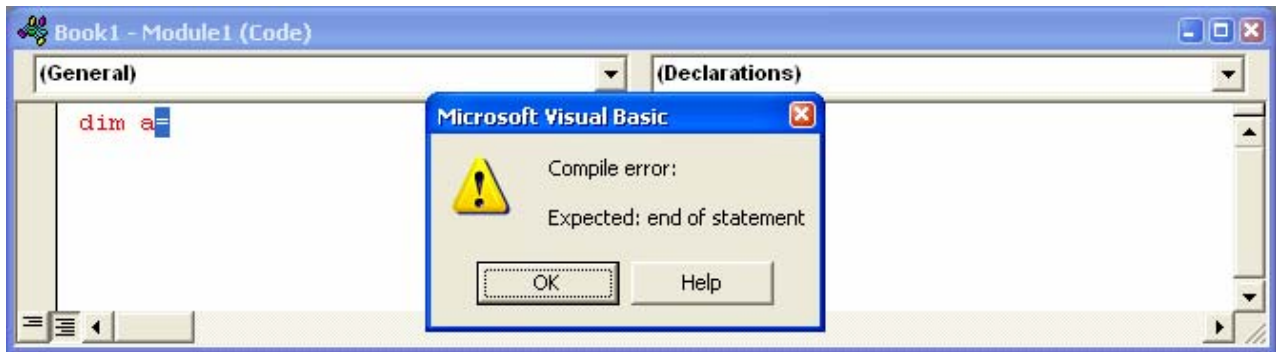
Các quy tắc về cú pháp thường khó nhớ đối với những người mới học lập trình hay mới sử dụng ngôn ngữ lập trình mới, cho nên, để thuận tiện cho người lập trình, VBA IDE cung cấp tính năng tự động phát hiện lỗi cú pháp trong quá trình viết mã lệnh. Tuy nhiên việc kiểm tra tự động này có thể gây khó chịu cho những lập trình viên chuyên nghiệp, những người rất hiếm khi mắc lỗi cú pháp khi lập trình, cho nên chức năng này chỉ hoạt động khi được kích hoạt, bằng cách chọn trình đơn **Tools** ⇒ **Options** ⇒ **Editor** ⇒ **Code Settings**.



Hình III-1: Bật / Tắt trợ giúp phát hiện lỗi cú pháp của VBA IDE

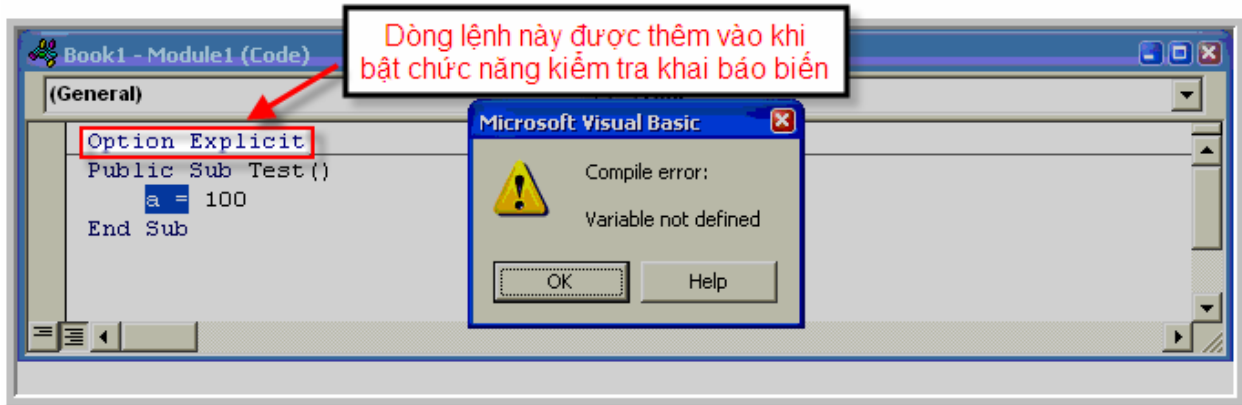
Ý nghĩa của hai tùy chọn này như sau:

- ✦ Tự động kiểm tra lỗi cú pháp (Auto Syntax Check): 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), một hộp thoại (như hình dưới đây) sẽ thông báo vị trí gây lỗi cũng như nguyên nhân gây lỗi. Nếu người dùng bỏ qua không sửa ngay thì dòng lệnh có lỗi sẽ được đánh dấu.



**Hình III-2: VBA IDE tự động kiểm tra lỗi cú pháp và thông báo cho người dùng**

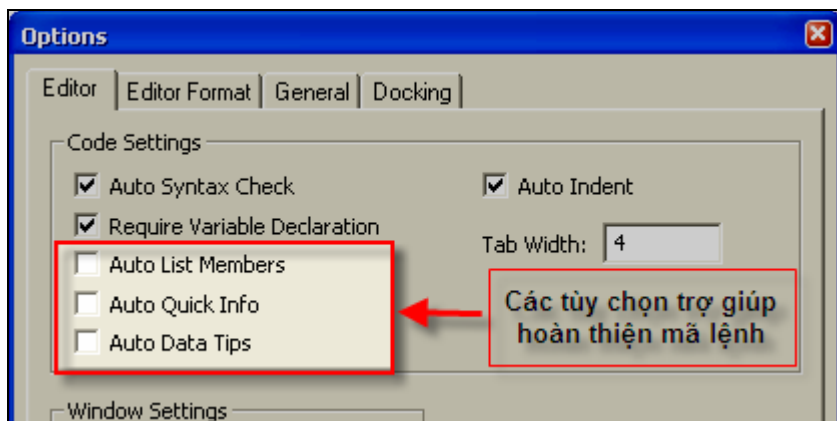
- ♦ Kiểm tra các biến (Require Variable Declaration): Trong VB, người dùng có thể sử dụng một biến mà không cần khai báo. Trong trường hợp này biến sẽ được khởi tạo và nhận một giá trị mặc định. Tuy nhiên, nếu lạm dụng điều này, rất có thể sẽ làm cho chương trình khó quản lý và dễ nhầm lẫn, vì thế VBA 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 sử dụng và VBA IDE sẽ tự động thêm vào đầu của mỗi mô-đun dòng lệnh “Option Explicit”.



**Hình III-3: VBA IDE tự động thông báo lỗi khi biến được sử dụng mà chưa khai báo**

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

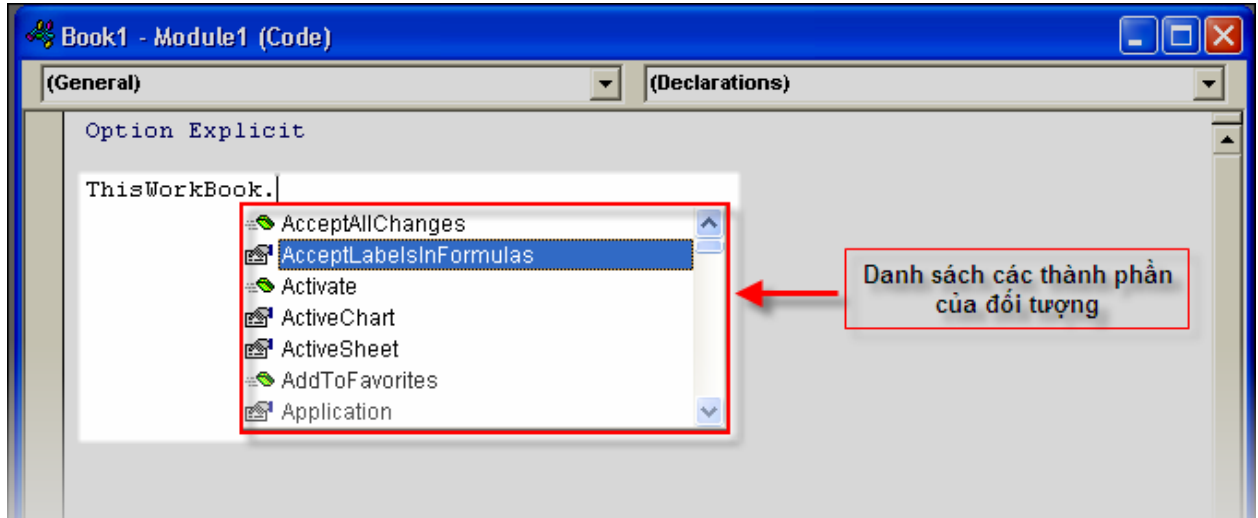
Mã lệnh, thông thường là một tập hợp bao gồm các từ khóa, câu lệnh, tên biến hay toán tử được sắp xếp theo một trật tự nhất định. Tên của các thành phần này có thể khó nhớ chính xác hoặc quá dài, cho nên VBA IDE đưa ra tính năng này bằng cách hiển thị những thành phần có thể phù hợp với vị trí dòng lệnh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



**Hình III-4: Bật / tắt trợ giúp hoàn thiện mã lệnh tự động trong VBA IDE**

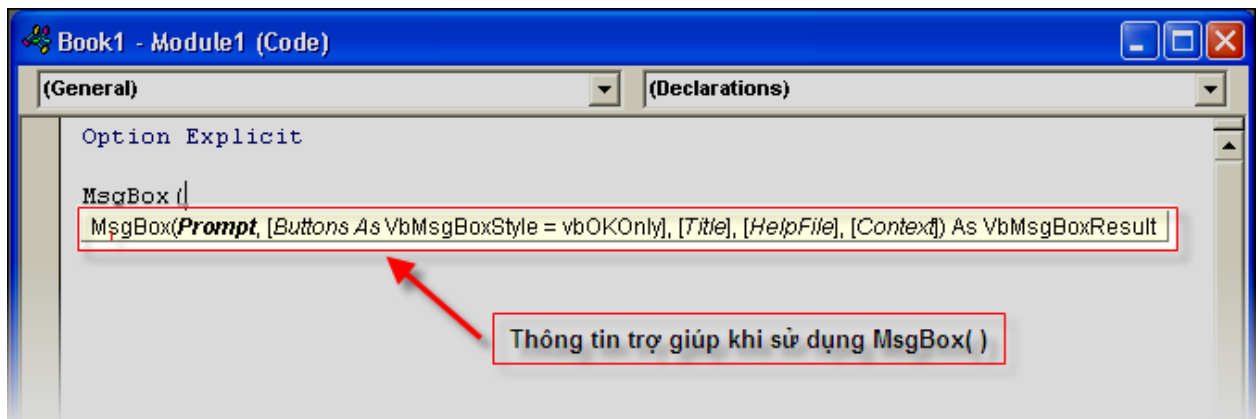
Ý nghĩa của các tùy chọn này như sau:

- ✦ Tự động hiển thị danh sách các thành phần của đối tượng (Auto List Member): 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 để người dùng chọn, sau khi bấm phím **Tab**, tên của thành phần này sẽ được tự động điền vào vị trí thích hợp trong dòng lệnh.



Hình III-5: Danh sách các thành phần được tự động hiển thị.

- ✦ Tự động hiển thị cú pháp cho chương trình con (Auto Quick Info): 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 sử dụng nó. Các thông tin này bao gồm tên của tham số cùng với kiểu của nó.



Hình III-6: Tự động hiển thị thông tin của các tham số trong chương trình con.

- ✦ Tự động hiển thị giá trị của biến (Auto Data Tips): 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.

Ngoài ra, nếu những tính năng trợ giúp trên chưa được kích hoạt, trong quá trình viết mã lệnh, người dùng có thể kích hoạt tạm thời chúng bằng cách nhấn tổ hợp phím **Ctrl + Space**. Cần chú ý rằng, khi danh sách trợ giúp hiện ra, người dùng có thể sử dụng chuột hoặc phím mũi tên để lựa chọn mục cần sử dụng trong danh sách đó rồi bấm phím **Tab** để xác nhận.

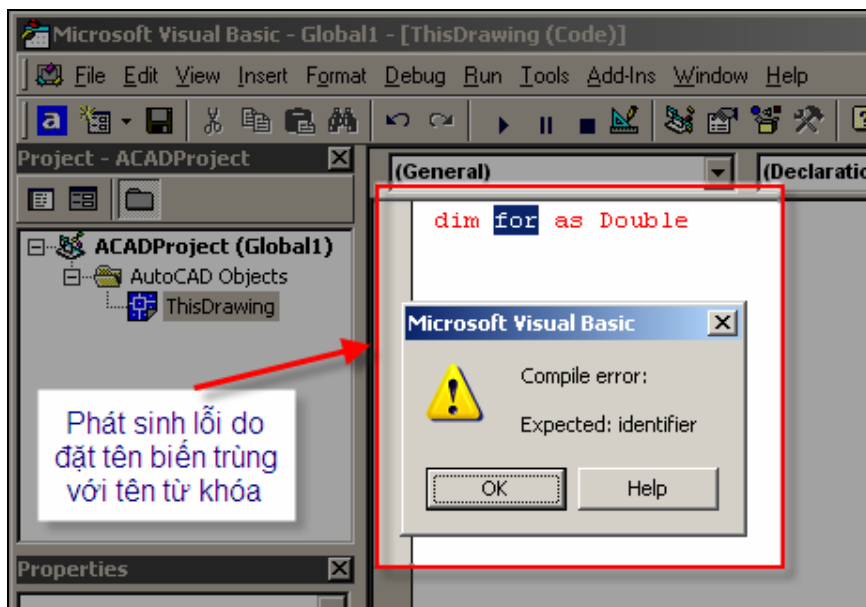
#### 4. Từ khoá trong VB

Từ khoá là tập hợp các từ cấu thành một ngôn ngữ lập trình. Mỗi ngôn ngữ lập trình đều có một bộ từ khoá riêng, dưới đây là danh sách các từ khoá trong ngôn ngữ lập trình VB:

## GIÁO TRÌNH TỰ ĐỘNG HOÁ THIẾT KẾ CẦU ĐƯỜNG

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
Else	Len	On	Seek	WithEvents
Empty	Let	Option	Set	
Error	Lock	Optional	Static	
False	Me	ParamArray	Step	

Các từ khóa là những từ được dùng riêng cho những chức năng khác nhau trong ngôn ngữ lập trình, ví dụ từ khóa “Private” hạn chế phạm vi sử dụng của biến hay chương trình con. Do đó việc đặt tên (biến, chương trình con) bắt buộc phải khác so với các từ khóa, nếu không sẽ phát sinh lỗi cú pháp.



Hình III-7: VBA IDE báo lỗi do tên biến trùng tên với từ khóa

### 5. Các kiểu dữ liệu cơ bản

Khi một chương trình vận hành, nó sẽ tác động và làm thay đổi giá trị của một vài thông số trong chương trình, ví dụ trong chương trình *giải phương trình bậc 2*, các thành phần trong phương trình:  $y=ax^2+bx+c$  sẽ cần thay đổi giá trị khi chương trình hoạt động. Như vậy giá trị của các thông số này có nhu cầu thay đổi trong những lần hoạt động khác nhau của chương trình cũng như trong một lần hoạt động nào đó, ví dụ giá trị của  $y$  sẽ thay đổi khi ta thay đổi giá trị của  $a$  trong phương trình trên. Chính bởi nhu cầu thay đổi giá trị này mà người ta đưa khái niệm “biến” để mô tả sự “động” của những thông số này. Với mỗi biến, giá trị của nó luôn được quy định là phải thuộc một *kiểu dữ liệu* nào đó, ví dụ giá trị của  $y$  trong phương trình trên phải là kiểu số thực. Do ngôn ngữ lập trình được thiết kế để thực hiện nhiều nhiệm vụ khác nhau cho nên trong ngôn ngữ lập trình nào cũng luôn có nhiều kiểu dữ liệu để thích ứng với nhu cầu đa dạng của việc lập trình.

Kiểu dữ liệu là loại giá trị mà một biến có thể nhận, nói cách khác, khi một biến được khai báo thì ta buộc phải gán cho nó một kiểu dữ liệu nhất định. Về tổng thể có thể chia các kiểu dữ liệu trong VB ra làm hai loại:

- ✦ Các kiểu dữ liệu được định nghĩa sẵn trong VB: là những kiểu dữ liệu cơ bản và thường gặp như kiểu số thực (Double), số nguyên (Integer), Chuỗi (String)...
- ✦ Các kiểu dữ liệu do người dùng tự định nghĩa: là kiểu dữ liệu được tự xây dựng dựa trên những thành phần dữ liệu cơ bản trong VB. Cách xây dựng kiểu dữ liệu này được đề cập trong phần dưới.

### 5.1. Kiểu logic (boolean)

Chỉ chứa hai giá trị TRUE và FALSE (đúng và sai). Khi chuyển từ các dữ liệu dạng số sang kiểu logic, 0 sẽ được chuyển thành FALSE còn giá trị khác sẽ được chuyển thành TRUE. Khi chuyển từ kiểu logic sang kiểu số, giá trị FALSE sẽ được chuyển thành 0 còn giá trị TRUE sẽ được chuyển thành -1.

```
\Khai báo biến A là kiểu logic
Dim A As Boolean
```

Biến A lúc này chỉ có thể nhận các giá trị: *True* hay *False*.

### 5.2. Kiểu số nguyên

Dùng để chứa các giá trị là số nguyên và có vài loại dữ liệu kiểu này. Sự khác nhau của những loại dữ liệu này là giới hạn giá trị (lớn nhất và nhỏ nhất) mà biến có thể nhận được (tham khảo bảng dưới).

Kiểu nguyên	số	Kích thước	Phạm vi
Byte		1 byte	0 đến 255
Integer		2 bytes	-32,768 đến 32,767
Long		4 bytes	-2,147,483,648 đến 2,147,483,647

### 5.3. Kiểu số thực

Dùng để chứa các giá trị là số thực. Các kiểu số thực thường dùng được trình bày trong bảng dưới đây:

Kiểu thực	số	K.thước	Phạm vi
Single		4 byte	Từ -3.402823E38 đến -1.401298E-45 và từ 1.401298E-45 đến 3.402823E38
Double		8 bytes	-1.79769313486231E308 đến -4.94065645841247E-324 và từ 4.94065645841247E-324 đến 1.79769313486232E308
Currency		8 bytes	Từ -922,337,203,685,477.5808 đến 922,337,203,685,477.5807

### 5.4. Kiểu mảng (array)

Khi gặp trường hợp phải xử lý một loạt các biến tương tự nhau, ví dụ các phần tử của một ma trận, nếu ta phải đặt tên khác nhau cho tất cả các biến này thì rất bất tiện, thay vào đó ta có thể dùng *kiểu mảng* để đặt tên chung cho cả nhóm các phần tử đó và khi nào cần sử dụng từng phần tử ta sẽ gọi tên theo *chỉ số* của chúng trong mảng.

```
\Khai báo mảng
Dim Matrix_1(10) As Double
```

Mảng Matrix\_1 trên có 11 phần tử liên tục được đánh số từ **0** đến **10** (ma trận có 1 hàng và 11 cột). Khi sử dụng ta chỉ việc gọi phần tử cần dùng theo chỉ số tương ứng.

```
`Gán giá trị 100 cho phần tử thứ 2  
Matrix_1(1)=100  
`Gán giá trị 100 cho phần tử cuối cùng  
Matrix_1(10)=100
```

Ta cũng có thể cố định phạm vi chỉ số của mảng bằng cách khai báo như sau:

```
`Khai báo mảng  
Dim Matrix_2(1 To 10) As Double
```

Lúc này chỉ số của mảng Matrix\_2 sẽ bắt đầu từ 1 và mảng này có 10 phần tử.

```
`Gán giá trị 200 cho phần tử thứ 2  
Matrix_2(2)=200  
`Gán giá trị 200 cho phần tử cuối cùng  
Matrix_2(10)=200
```

Ví dụ sau khai báo và sử dụng (gán giá trị cho phần tử) một ma trận 3 hàng 5 cột

```
`Khai báo mảng (3x5)  
Dim Matrix_3(1 To 3, 1 To 5) As Double  
`Gán giá trị 100 cho phần tử tại hàng thứ 2 cột thứ 3  
Matrix_3(2,3)=100
```

Trong VB, mảng có thể có một chiều hoặc nhiều chiều, kích thước của mảng được xác định dựa trên số chiều và biên trên, biên dưới của mỗi chiều. Các thành phần trong mảng là liên tục giữa hai biên.

Trong các ví dụ trên, các mảng có kích thước (hay số lượng phần tử) là không thay đổi trong suốt quá trình hoạt động của chương trình. Người ta gọi loại mảng này là *mảng tĩnh* và thường được dùng cho những bài toán biết trước số phần tử của mảng hay kích thước mảng không lớn. Ngoài loại mảng tĩnh này, trong VB còn cho phép định nghĩa một loại mảng khác mà kích thước (hay số lượng phần tử) của nó có thể thiết lập lại ngay trong lúc chương trình đang hoạt động, người ta gọi loại mảng này là *mảng động*. Với mảng động, người lập trình không cần biết số phần tử của mảng trong lúc lập trình, số phần tử này sẽ được thiết lập trong quá trình chương trình hoạt động dựa theo nhu cầu của từng bài toán cụ thể.

Khi một mảng động, mà các phần tử của nó đã được gán giá trị, cần thay đổi kích thước, sẽ có hai tình huống cần xét đến:

- ✦ Toàn bộ giá trị ban đầu (trước lúc thay đổi kích thước mảng) sẽ bị hủy bỏ, các phần tử mảng mới (sau khi thay đổi kích thước) sẽ nhận giá trị mặc định.

```
` Khai báo mảng A là mảng động  
Dim A() As Long  
` Xác định kích thước cho mảng động A: mảng 1 chiều có 5 phần tử  
Redim A(1 to 5) As Long  
` Gán giá trị cho phần tử của mảng A  
A(1) = 100: A(2) = 200  
` Định lại kích thước cho mảng A: mảng hai chiều với 3x3=9 phần tử  
Redim A(1 to 3, 2 to 4) as Long
```

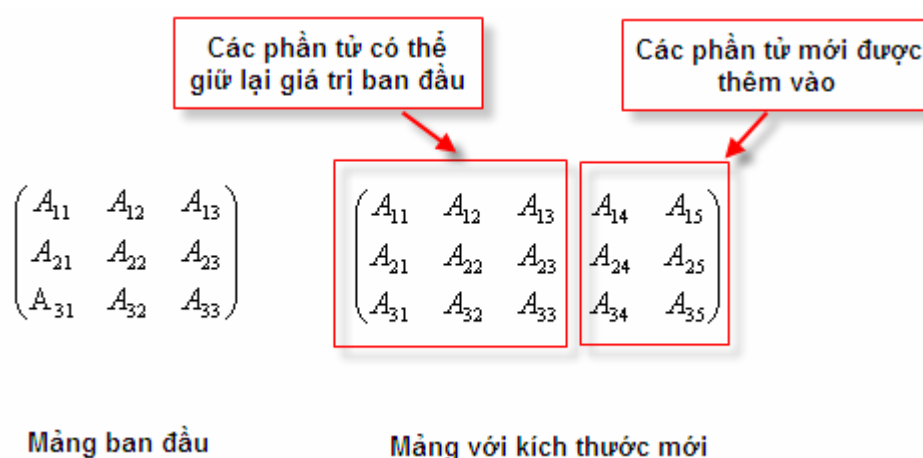
Sau dòng cuối cùng này, toàn bộ giá trị của mảng A cũ (có A[1]=100 và A[2]=200) sẽ bị xóa bỏ và tất cả các phần tử mới của mảng A (9 phần tử) sẽ nhận giá trị mặc định (thường được gán bằng 0).

- ✦ Giá trị cũ của các phần tử mảng sẽ được giữ lại khi cả hai điều kiện sau thỏa mãn:

- Sử dụng lệnh `ReDim` với từ khóa `Preserve`.
- Sự thay đổi kích thước mảng chỉ được thực hiện ở biên trên của chiều cuối cùng của mảng, nghĩa là các phần tử cần giữ lại giá trị có chỉ số không đổi ngay cả khi mảng được định lại kích thước.

```

`Khai báo mảng động A
Dim A() As Long
`Gán kích thước cho mảng A
ReDim A(1 To 3, 1 To 3) As Long
`Gán giá trị cho phần tử của mảng A
A(1,1) = 100: A(1,2) = 200
A(2,1) = 150: A(2,2) = 250
`Định lại kích thước cho mảng A, giữ lại giá trị ban đầu
`của các phần tử, lưu ý đến phạm vi của mảng mới
ReDim Preserve A(1 To 3, 1 To 5) As Long
    
```



Hình III-8: Các phần tử có thể giữ lại giá trị ban đầu và các phạm vi có thể thay đổi kích thước của mảng động

Trong ví dụ trên, các phần tử của mảng A được giữ lại giá trị sau khi kích thước của mảng được thay đổi lại. Lưu ý, ta chỉ có thể giữ lại giá trị của mảng ban đầu khi sự mở rộng được thực hiện ra biên cuối cùng của nó như hình trên.

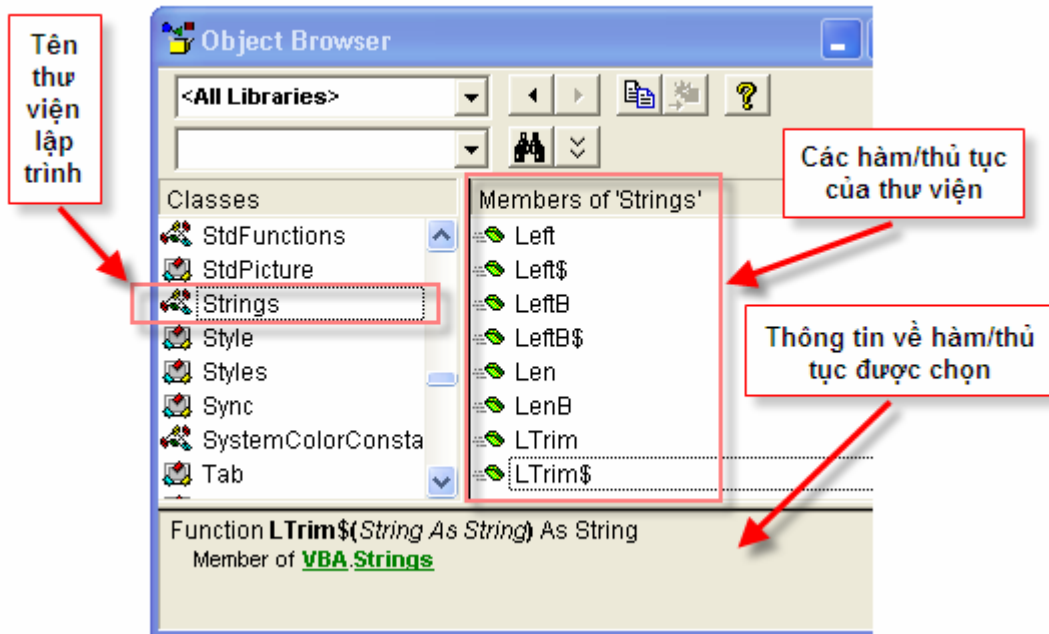
### 5.5. Kiểu chuỗi (String)

Chuỗi là một hàng bao gồm các ký tự liên tục nhau, các ký tự ở đây rất đa dạng: có thể là chữ số, chữ cái, dấu cách (space), ký hiệu. Số lượng ký tự trong một chuỗi là rất lớn ( $2^{16}$  ký tự). Mặc định trong VB, các biến hay tham số kiểu chuỗi có chiều dài thay đổi tùy theo giá trị dữ liệu được gán cho nó.

```

Dim S As String
S="ABCD 1234 @#$%"
    
```

Để tạo điều kiện thuận lợi cho người dùng, bên trong VB có sẵn một số hàm liên quan đến xử lý chuỗi, ví dụ như cắt chuỗi, tách chuỗi, ghép chuỗi, tìm kiếm, ... Các hàm cơ bản này được trình bày ở phần sau trong giáo trình này hoặc có thể tra cứu toàn bộ các hàm liên quan trong MSDN (Microsoft Developer Network) hoặc Object Browser (thư viện Strings) bằng cách nhấn phím F2 trong giao diện lập trình VBA IDE



Hình III-9: Thông tin về các hàm trong thư viện lập trình của VBA được hiển thị trong Object Browser

### 5.6. Kiểu thời gian (Date)

Dùng để lưu trữ và thao tác trên các giá trị thời gian (ngày và giờ). Định dạng ngày và giờ phụ thuộc vào các thiết lập về hiển thị trong hệ thống của người dùng. Khi chuyển từ các dữ liệu kiểu số sang kiểu ngày tháng, các giá trị ở bên trái dấu phẩy chuyển thành thông tin về ngày còn giá trị ở bên phải dấu phẩy sẽ được chuyển thành thông tin về giờ.

```
Dim D As Date
Dim S As String
D = Now()
S = "Ngày: " & Day(D) & " - Tháng: " & Month(D) & " - Năm: " & Year(D)
Debug.Print (S)
```

Ví dụ trên sẽ hiển thị thông tin về thời gian (ngày – tháng – năm) trong cửa sổ Immediate của VBA IDE.

### 5.7. Kiểu Variant

Kiểu Variant là một kiểu dữ liệu đặc biệt có thể chứa tất cả các loại dữ liệu, ngoại trừ kiểu chuỗi có chiều dài cố định. Kiểu Variant cũng có thể chứa các giá trị đặc biệt như Empty, Error, Nothing và Null.

Tuy kiểu dữ liệu Variant có vẻ tiện dụng nhưng khi sử dụng một cách quá thoải mái thì nguy cơ gây lỗi của loại biến này là rất lớn, đặc biệt khi thao tác với các toán tử.

```
Dim V As Variant
\ Gán biến V với một chuỗi
V = "String"
\ Gán biến V với một số
V = 16
\ Gán biến V với giá trị kiểu logic
V = True
\ Gán biến V với một dữ liệu kiểu thời gian
V = #01/06/2007#
```



Sử dụng hàm VarType(vVariant) sẽ cho ta mã của kiểu dữ liệu hiện đang lưu trữ trong biến Variant.

Giá trị VarType	Chú thích
<b>0-vbEmpty</b>	Không có gì trong variant
<b>1-vbNull</b>	Không có dữ liệu hợp lệ trong variant
<b>2-vbInteger</b>	Variant chứa Integer
<b>4-vbSingle</b>	Variant chứa Single
<b>7-vbDate</b>	Variant chứa Date/Time
<b>8-vbString</b>	Variant chứa String
<b>9-vbObject</b>	Variant chứa một Object
<b>11-vbBoolean</b>	Variant chứa Boolean

### 5.8. Kiểu tự định nghĩa (user-defined type)

Kiểu tự định nghĩa là kiểu dữ liệu do người dùng định nghĩa, tương tự như kiểu bản ghi (Record) trong ngôn ngữ lập trình Pascal hay kiểu cấu trúc (Struct) trong ngôn ngữ lập trình C. Kiểu tự định nghĩa bao gồm nhiều trường dữ liệu, mỗi trường dữ liệu có thể là các kiểu dữ liệu cơ bản hoặc các kiểu tự định nghĩa khác.

Ví dụ, khi đo toàn đạc bằng máy kinh vĩ cơ, với mỗi điểm đo ta cần lưu lại các thông tin sau:

Ký hiệu	Ý nghĩa	Kiểu giá trị
<b>TrM</b>	Số hiệu trạm đặt máy	Integer
<b>STT</b>	Thứ tự của điểm đo	Integer
<b>DT</b>	Số đọc dây trên	Double
<b>DG</b>	Số đọc dây giữa	Double
<b>DD</b>	Số đọc dây dưới	Double
<b>H</b>	Góc bằng	Double
<b>V</b>	Góc đứng	Double
<b>MT</b>	Mô tả đặc điểm của điểm đo	String

Với một chương trình xử lý số liệu đo toàn đạc, cách tốt nhất là quản lý theo điểm đo, và do đó mỗi điểm đo là một biến có kiểu dữ liệu phù hợp với bảng trên. Đó chính là kiểu dữ liệu tự định nghĩa.

```

`Định nghĩa kiểu dữ liệu cho điểm đo toàn đạc
Type DiemDo
    TrM As Integer
    STT As Integer
    DT As Double
    DG As Double
    DD As Double
    H As Double
    V As Double
    MT As String
End Type
    
```

Sau khi định nghĩa kiểu dữ liệu DiemDo xong, ta có thể sử dụng nó như những kiểu dữ liệu thông thường khác.

```
'Khai báo biến sử dụng kiểu dữ liệu tự định nghĩa
Dim P1 As DiemDo
Dim P_Array(1 to 1000) As DiemDo
With P1
    .TrM = 1
    .STT = 1
    .DT = 2130
    .DG = 2120
    .DD = 2110
    .H = 130.5
    .V = 78.25
    .MT = "Goc nha C4"
End With
P_Array(1) = P1
```

Từ khóa: **With ... End With** dùng để tránh phải nhập lại nhiều lần tên biến kiểu dữ liệu tự định nghĩa. Dấu chấm (.) được sử dụng để thao tác với các thành phần bên trong của biến có kiểu dữ liệu tự định nghĩa. Ví dụ sau là tương đương với ví dụ trên, nhưng không sử dụng cặp từ khóa **With ... End With**, chú ý là dấu chấm (.) luôn có:

```
'Khai báo biến sử dụng kiểu dữ liệu tự định nghĩa
Dim P1 As DiemDo
Dim P_Array(1 to 1000) As DiemDo
P1.TrM = 1
P1.STT = 1
P1.DT = 2130
P1.DG = 2120
P1.DD = 2110
P1.H = 130.5
P1.V = 78.25
P1.MT = "Goc nha C4"
P_Array(1) = P1
```

### 5.9. Kiểu lớp (Class)

Kiểu lớp (Class) là một mở rộng của kiểu dữ liệu tự định nghĩa, sự khác biệt cơ bản ở đây là trong kiểu lớp còn có những đoạn chương trình dùng để xử lý chính những dữ liệu trong nó. Dữ liệu bên trong lớp thường được gọi là các thuộc tính (Properties), còn những đoạn chương trình trong lớp để xử lý dữ liệu này thực chất là các Hàm / Thủ tục (Function / Sub) được định nghĩa bên trong lớp và thường được gọi là các Phương thức (Methods). Một biến có kiểu dữ liệu là lớp được gọi là một đối tượng (Object) và cách sử dụng các Properties, Methods của đối tượng này tương tự như cách sử dụng các thành phần của kiểu dữ liệu tự định nghĩa.

Lớp cần được xây dựng trong Class Module hoặc ta có thể sử dụng lại các lớp sẵn có từ các thư viện lập trình.

Như vậy lớp có thể gồm các thành phần sau:

- ✦ Các thuộc tính (Property): là các dữ liệu mô tả trạng thái của bản thân đối tượng hoặc các quan hệ của nó với các đối tượng khác. Về bản chất, thuộc tính là các biến được khai báo trong lớp đó. Kiểu dữ liệu của các thuộc tính có thể là các kiểu dữ liệu cơ bản hoặc có thể là một lớp khác (kiểu Class).

- ✦ Các phương thức (Method): mô tả hành vi, chức năng của đối tượng. Về bản chất, phương thức là các chương trình con được xây dựng bên trong lớp và chúng có nhiệm vụ xử lý các dữ liệu của chính lớp đó.
- ✦ Các sự kiện (Event): Sự kiện giúp cho lớp có khả năng giao tiếp với các lớp khác hoặc với môi trường ngoài.

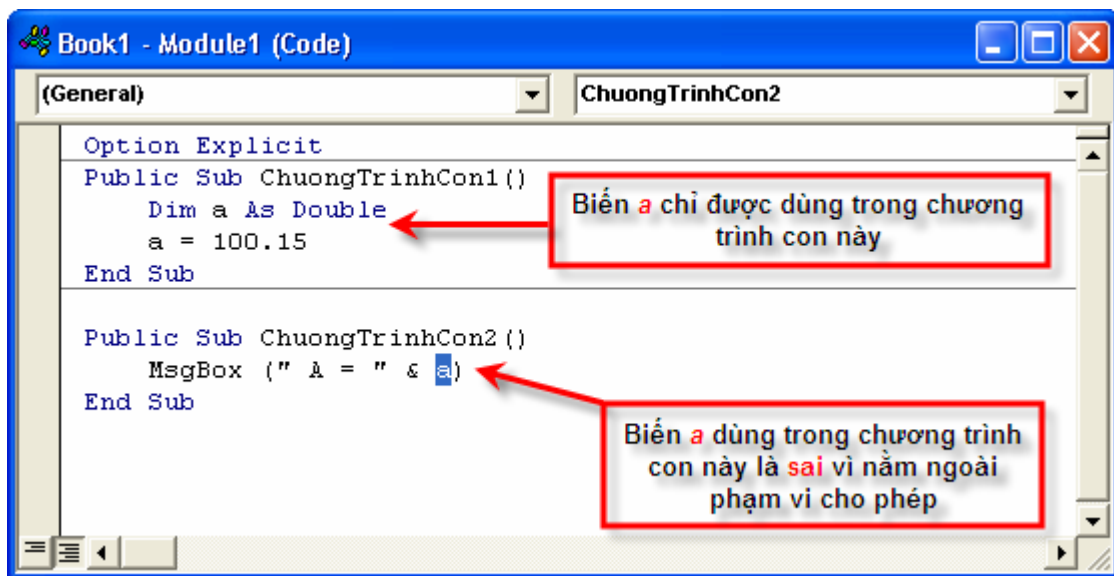
Trong khuôn khổ của giáo trình này, các vấn đề liên quan đến xây dựng lớp sẽ không được đề cập chi tiết. Tuy nhiên nếu ai quan tâm có thể tìm hiểu thêm trong giáo trình “Lập trình hướng đối tượng trong xây dựng” của bộ môn Tự động hóa thiết kế Cầu đường.

## 6. Khai báo biến trong VB

Trong VB, muốn sử dụng một biến có thể không cần khai báo, tuy nhiên cách làm này chỉ nên dùng khi viết các chương trình nhỏ, còn đối với các chương trình lớn, có nhiều mô-đun, thì nên bắt buộc khai báo biến trước khi sử dụng (theo cách thiết lập ở mục 2 của chương này).

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 trên những nguyên tắc sau:

- ✦ Khi biến khai báo trong chương trình con nào thì phạm vi sử dụng của nó được giới hạn trong chính chương trình con đó. Biến loại này được gọi là biến *cục bộ*.



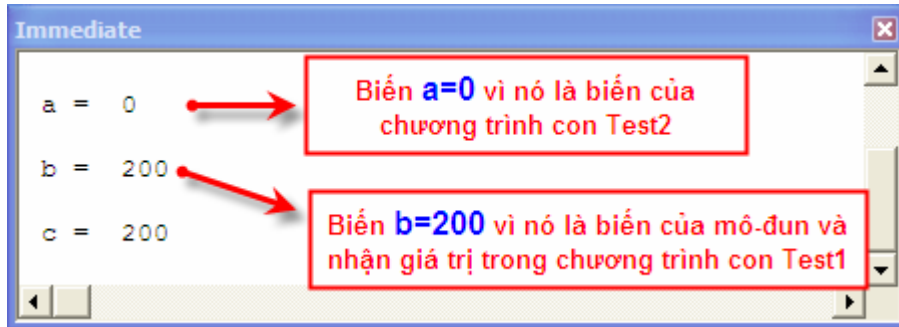
Hình III-10: Phạm vi sử dụng của biến được khai báo trong chương trình con

- ✦ Nếu biến được khai báo ở cấp mô-đun và biến được khai báo trong chương trình con có tên trùng nhau thì ở bên trong chương trình con, biến được sử dụng là biến được khai báo bên trong nó. Ta xét ví dụ sau:

```
Option Explicit
Dim a As Double, b As Double
Public Sub Test1()
    a = 100 : b = 200
End Sub
Public Sub Test2()
    Dim a As Double, c As Double
    Test1
    c = a + b
    Debug.Print "a = "; a;
    Debug.Print "b = "; b;
```

```
Debug.Print "c = "; c;
End Sub
```

Biến a và b được khai báo ở cấp mô-đun, nghĩa là mọi chương trình con trong mô-đun này đều có thể sử dụng và tác động lên chúng. Giá trị của a và b được gán trong chương trình con Test1. Trong chương trình con Test2 một biến a khác được khai báo (trùng tên với biến a của mô-đun), và giá trị khởi tạo của nó bằng 0. Kết quả chạy chương trình con Test2 như sau:

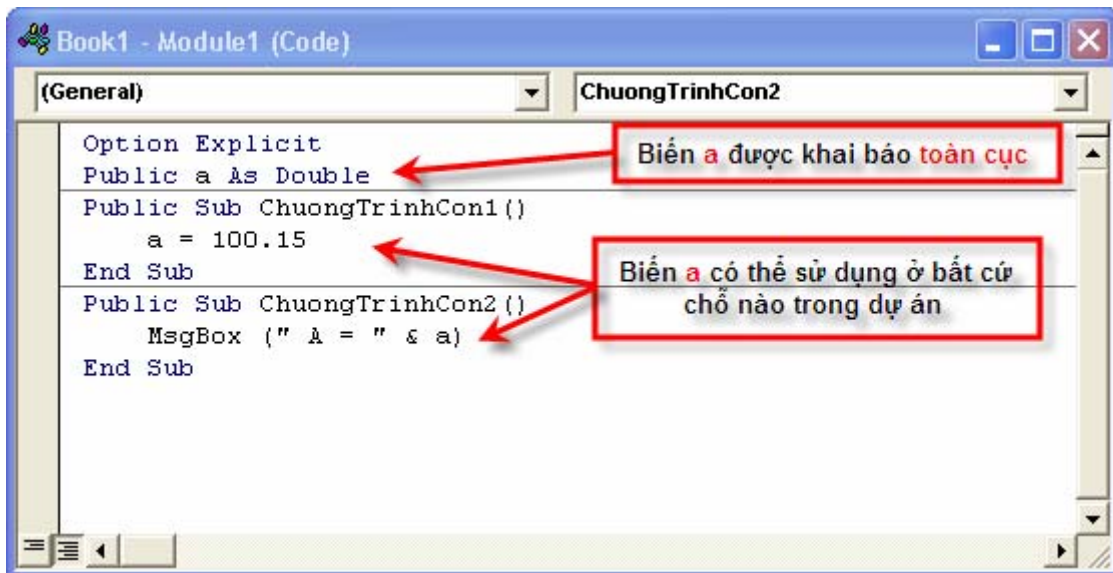


**Hình III-11: Mức độ ưu tiên trong sử dụng biến**

- ✦ Sử dụng từ khóa `Public` để xác định phạm vi sử dụng biến là trong toàn bộ dự án, nghĩa là từ bất cứ nơi đâu trong dự án (mô-đun chuẩn, mô-đun lớp, và Userform) đều có thể sử dụng biến này. Biến được khai báo với từ khóa `Public` thường được gọi là biến *toàn cục*. Trong mô-đun nào đó, nếu một biến được khai báo với từ khóa `Dim`, thì mặc định, biến đó là biến cục bộ, nghĩa là tương đương với việc sử dụng từ khóa `Private`.



**CHÚ Ý** Không sử dụng các từ khoá `Public`, `Private` hay `Friend` cho khai báo dữ liệu nằm bên trong chương trình con.

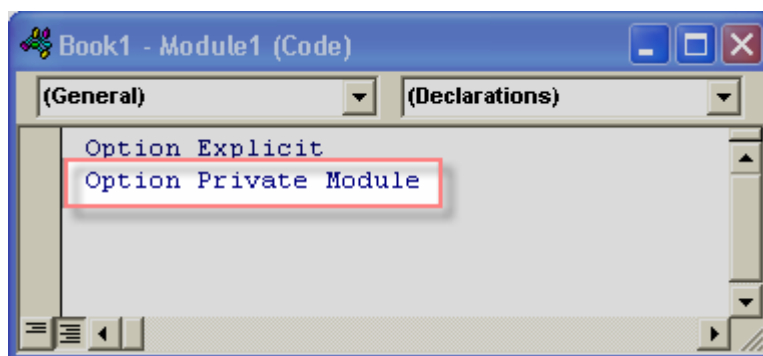


**Hình III-12: Phạm vi sử dụng biến toàn cục**

Ở mức độ rộng hơn, có thể coi biến như một khối dữ liệu của chương trình và mức độ toàn cục được chia làm hai loại như sau:

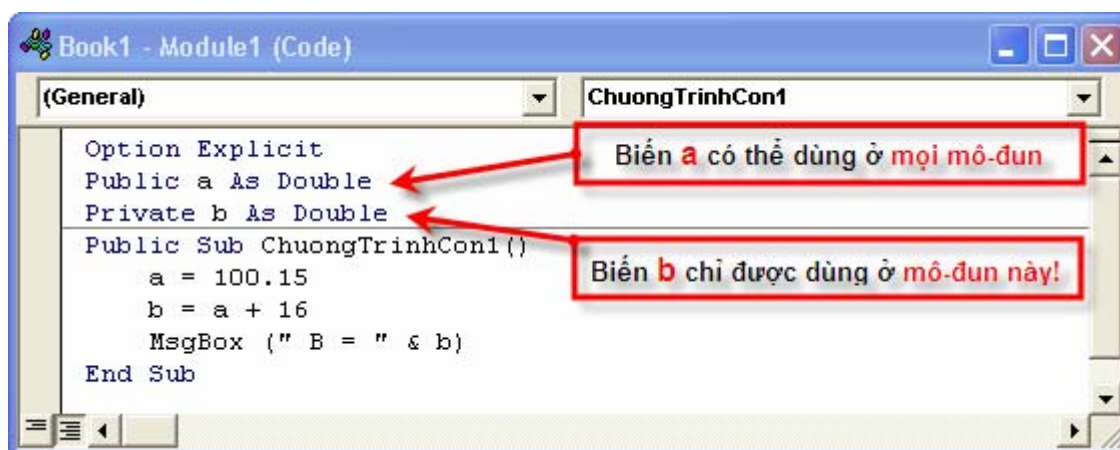
- Toàn cục ở mức ứng dụng: Trong trường hợp ứng dụng gồm nhiều dự án (multi-projects), nếu trong một mô-đun **không có** khai báo lựa chọn `Option Private Module` thì tất cả các thành phần dữ liệu hay chương trình được khai báo `Public` trong mô-đun đó có phạm vi hoạt động toàn bộ ứng dụng – nghĩa là chúng còn có thể được tham chiếu từ những dự án khác trong ứng dụng.

- Toàn cục ở mức dự án: Trong trường hợp ứng dụng gồm nhiều dự án (multi-projects), nếu trong một mô-đun có khai báo lựa chọn Option Private Module thì tất cả các thành phần dữ liệu hay chương trình được khai báo Public trong mô-đun đó chỉ có phạm vi hoạt động trong nội bộ dự án chứa mô-đun mà không thể được tham chiếu từ những dự án khác trong ứng dụng.



Hình III-13: Khai báo tùy chọn phạm vi biến ở mức dự án.

- ◆ Sử dụng từ khóa Private để xác định phạm vi hoạt động của biến là trong nội bộ của mô-đun đó, tất cả các chương trình con hay bất cứ thành phần nào của mô-đun này đều có thể sử dụng biến loại này nhưng chúng không thể truy cập được từ những mô-đun hay Userform khác trong dự án.



Hình III-14: Phạm vi sử dụng của biến tương ứng với từ khóa Public và Private.

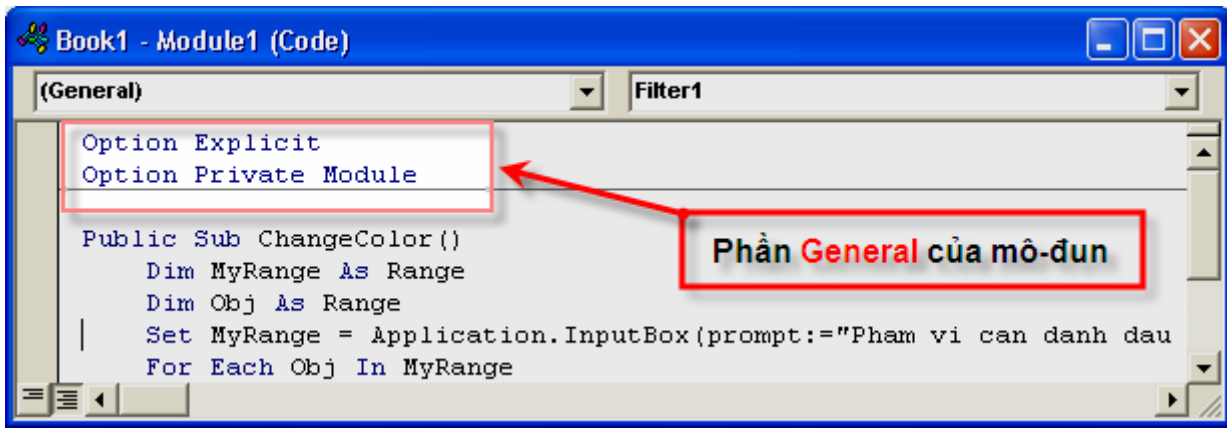
**CHÚ Ý** Khi khai báo kiểu dữ liệu người dùng tự định nghĩa hoặc các chương trình con trong một mô-đun, nếu không chỉ rõ phạm vi hoạt động thì phạm vi hoạt động mặc định là **Public**.

Để tránh các nhầm lẫn do không nhớ phạm vi hoạt động mặc định, người dùng nên chỉ rõ phạm vi hoạt động của chương trình hay dữ liệu ngay khi khai báo.

- ◆ Ngoài ra, trong các mô-đun lớp (Class Module) hoặc mô-đun lệnh của UserForm còn có thể sử dụng từ khóa Friend để xác định phạm vi hoạt động của một chương trình con (phương thức). Khi sử dụng từ khóa này, chương trình con có thể được truy xuất từ mọi nơi trong nội bộ dự án (Project) chứa nó nhưng không thể được truy xuất trong những dự án khác của ứng dụng (khác với khi dùng từ khóa Public – chương trình con có thể được truy xuất từ mọi nơi của ứng dụng).

**CHÚ Ý** Các khai báo dữ liệu với các từ khoá trên được thực hiện trong phần General của một mô-đun. Các dữ liệu đó còn được gọi là dữ liệu cấp mô-đun (module level).

Trong mỗi mô-đun, phần đầu tiên (của phần viết mã lệnh) được gọi là phần General của mô-đun đó. Theo quy ước, các thiết lập cho mô-đun được đặt ở đây và VBA IDE sẽ tự động phân cách phần này. Không có giới hạn về kích thước cho phần này.



Hình III-15: Phần General trong mô-đun

### 6.1. Khai báo hằng số

Hằng số là một loại biến đặc biệt mà giá trị của nó được xác định ngay lúc khai báo và luôn không thay đổi. Ta nên dùng cách này cho những hằng số hay phải dùng lặp lại trong chương trình, ví dụ như hằng số  $\pi = 3.14159$ . Sau khi khai báo hằng số này:

```
Const Pi=3.14159
```

ta luôn có thể sử dụng giá trị 3.14159 bất cứ chỗ nào trong chương trình với cái tên dễ nhớ hơn là Pi.

Cú pháp:

```
[Public/ Private] Const <tên_hằng>=<giá_trị_hằng_số>
```

Các từ khoá `Public` hay `Private` xác định phạm vi hiệu lực của hằng số, với từ khoá `Public`, hằng số này có thể sử dụng ở bất cứ đâu trong ứng dụng, còn với từ khoá `Private` thì hằng số này chỉ có thể sử dụng bên trong mô-đun nơi khai báo hằng số đó. Ý nghĩa của hai từ khoá này cũng không thay đổi cho tất cả các phần khác mà có sử dụng chúng.

### 6.2. Khai báo biến

Cú pháp:

```
Dim <tên_biến> as <Kiểu_dữ_liệu>
```

Khi dùng từ khóa `Public` hay `Private` nhằm xác định phạm vi hiệu lực của biến thay cho từ khóa `Dim` trong khai báo biến thì cú pháp như sau:

```
Public <tên_biến> as <Kiểu_dữ_liệu>
```

Hay:

```
Private <tên_biến> as <Kiểu_dữ_liệu>
```

### 6.3. Khai báo kiểu tự định nghĩa

Trong VB có thể khai báo các kiểu dữ liệu theo nhu cầu của người sử dụng. Cú pháp khai báo như sau:

```
Type <Tên_kiểu>
    <tên_trường_1> as <Kiểu_dữ_liệu>
    <tên_trường_2> as <Kiểu_dữ_liệu>
    ...
    <tên_trường_n> as <Kiểu_dữ_liệu>
End Type
```

Sau khi khai báo kiểu tự định nghĩa, người dùng có thể sử dụng các biến có kiểu tự định nghĩa bằng cách khai báo như các biến thông thường, với <Kiểu\_dữ\_liệu> được thay bằng <Tên\_kiểu>. Để truy cập tới một trường của biến kiểu bản ghi, dùng toán tử (.) hoặc dùng cặp từ khóa With... End With.

**CHÚ Ý** Các từ khoá *Public* hay *Private* nhằm xác định phạm vi hoạt động của kiểu dữ liệu được khai báo. Đồng thời khai báo kiểu chỉ được thực hiện ở cấp mô-đun (không thực hiện được trong các chương trình con). Khi không chỉ rõ thì phạm vi hoạt động thì mặc định của một kiểu dữ liệu tự định nghĩa là *Public*.

## 6.4. Khai báo mảng tĩnh

Cú pháp:

```
[Public/Private/Dim] <tên_mảng> (<thông_số_về_chiều>) as <tên_kiểu>
```

Các thông số về chiều có thể biểu diễn qua các ví dụ sau:

Dim a(3 To 5) As Integer ` Mảng 1 chiều với các chỉ số từ 3 đến 5

Dim A(3) As Long ` Mảng 1 chiều với chỉ số đến 3 (mảng 1 chiều có 4 phần tử với chỉ số từ 0 đến 3)

Dim A(2 To 4, 6) As Double ` Mảng 2 chiều với một miền chỉ số từ 2 tới 4 và một miền có chỉ số từ 0 đến 6.

**GỢI Ý** Các từ khoá *Public* hay *Private* xác định phạm vi hoạt động của biến mảng (trong trường hợp mảng được khai báo mức mô-đun). Các qui định về phạm vi hoạt động của mảng tương tự với biến thông thường - đã được trình bày ở phần trước.

## 6.5. Khai báo mảng động

Cú pháp:

```
[Public/ Private/ Dim] <tên_mảng> () as <tên_kiểu>
```

Trong khai báo trên không chứa các thông số về chiều và đó thuần túy chỉ là một khai báo. Các phần tử của mảng chưa được tạo ra (hay nói cách khác mảng vẫn chưa thực sự được cấp phát bộ nhớ) và vẫn chưa sẵn sàng để sử dụng. Trước khi sử dụng mảng động hoặc khi muốn thay đổi kích thước của mảng, sử dụng lệnh *ReDim*. Cú pháp như sau:

```
Redim <tên_mảng> (<các thông số về chiều>) as <tên kiểu>
```

Chú ý rằng <tên\_kiểu> phải đúng như khai báo ban đầu, các thông số về chiều có thể khác trước cả về số chiều và kích thước của từng chiều. Khi đó, các dữ liệu cũ trong mảng không còn nữa, thay vào đó là những phần tử mới được khởi tạo.

## 6.6. Khai báo, tạo và làm việc với biến đối tượng

Khai báo và tạo biến đối tượng phải dùng thêm từ khóa `New`

```
Dim <tên_biến> as New <Kiểu_dữ_liệu>
```

<Kiểu\_dữ\_liệu> là lớp (class) đã được định nghĩa từ trước.

Phép gán đối tượng được thực hiện với từ khóa `Set`

```
Set <biến_đối_tượng> = <giá_trị>
```

Chú ý rằng nếu thực hiện khai báo một biến đối tượng như thông thường (không có từ khóa `New`) thì biến thực sự chưa được tạo ra. Trong trường hợp đó, người sử dụng phải tạo và gán đối tượng với các từ khóa tương ứng là `New` và `Set`.

```
Dim <tên_biến> as <Kiểu_dữ_liệu>  
Set <tên_biến> = New <Kiểu_dữ_liệu>
```

**CHÚ Ý** Câu lệnh `Set` **không phải là câu lệnh khai báo**, vì vậy nó phải được viết trong một chương trình con nào đó chứ không thể nằm trong phần `General` của một mô-đun.

Làm việc với một biến đối tượng tức là quá trình thao tác với đối tượng thông qua các thuộc tính, phương thức và các sự kiện của đối tượng đó. Để truy cập tới các thuộc tính và phương thức của đối tượng ta sử dụng theo cú pháp sau, chú ý đến dấu chấm ( `.` ) giữa tên biến và tên thuộc tính hay tên phương thức:

```
<Tên_biến>.<Tên_thuộc_tính>  
<Tên_biến>.<Tên_phương_thức> <(tham_số_của_phương_thức)>
```

## 7. Các toán tử và hàm thông dụng

### 7.1. Các toán tử

Toán tử được sử dụng cho mục đích xử lý dữ liệu. Ta sử dụng các toán tử để thực hiện tính toán, so sánh, gán và thực hiện nhiều thao tác khác. Dưới đây là danh sách và ý nghĩa của một số toán tử thông dụng:

Toán tử	Mô tả
Toán tử gán	
=	Gán giá trị cho biến hoặc thuộc tính
Toán tử toán học	
+	Cộng



-	Trừ
*	Nhân
/	Chia
\	Chia lấy phần nguyên
Mod	Chia lấy phần dư
^	Luỹ thừa
Toán tử logic	
Not	Trả về giá trị phủ định với giá trị biểu thức. Not(TRUE)=FALSE
And	Nối logic hai biểu thức. (TRUE And TRUE)=TRUE; các trường hợp khác cho kết quả bằng FALSE
Or	(FALSE or FALSE)=FALSE; các trường hợp khác cho kết quả là TRUE
Xor	Cho kết quả TRUE nếu hai đối số có cùng giá trị; ngược lại cho kết quả là FALSE
Eqv	So sánh hai giá trị logic; cách thức xử lý tương tự như toán tử Xor
Toán tử so sánh	
=	So sánh bằng
<>	Khác nhau
>	Lớn hơn
>=	Lớn hơn hoặc bằng
<	Nhỏ hơn
<=	Nhỏ hơn hoặc bằng

## 7.2. 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:

Hàm	Mô tả
Abs(x)	Lấy giá trị tuyệt đối
Exp(x)	Lấy mũ cơ số tự nhiên
Log(x)	Logarit cơ số tự nhiên
Sqr(x)	Lấy bình phương
Cos(x), Sin(x), Tan(x)	Hàm lượng giác
Atn(x)	Hàm lượng giác ngược
Fix(x)	Lấy phần nguyên (trước dấu phẩy). Fix(3.7)=3
Int(x)	Lấy phần nguyên đã được làm tròn. Int(3.7)=4
Round(x,num)	Làm tròn số thực <x> đến <num> chữ số sau dấu phẩy
Val(str)	Chuyển đổi chuỗi <str> thành giá trị kiểu số

## 7.3. Các hàm chuyển đổi dữ liệu

Chuyển đổi định dạng số liệu là một nhu cầu thường gặp trong lập trình do các ngôn ngữ lập trình luôn đòi hỏi kiểu dữ liệu phải rõ ràng và cố định cho từng biến nhằm tránh phát sinh các lỗi sau này. Việc chuyển đổi này, nếu trong trường hợp thông thường, thì VB sẽ tự động thực

## GIÁO TRÌNH TỰ ĐỘNG HOÁ THIẾT KẾ CẦU ĐƯỜNG

hiện. Nhưng khi gặp các yêu cầu đặc biệt thì buộc người dùng phải sử dụng những hàm chuyển đổi phù hợp.

**CHÚ Ý** Việc chuyển đổi kiểu dữ liệu luôn có thể tạo ra lỗi do không thể chuyển đổi được hoặc phát sinh kết quả sai. Cho nên khi sử dụng cần chú ý đến các khả năng gây lỗi của việc chuyển đổi kiểu dữ liệu.

Các hàm 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:

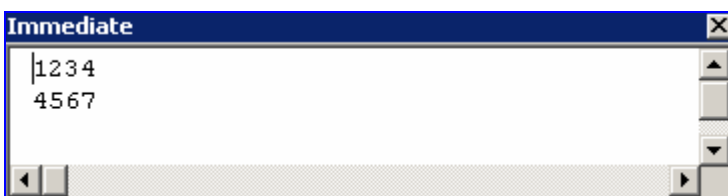
Hàm	Mô tả
<code>CBool(Expression)</code>	Chuyển đổi dữ liệu sang kiểu logic (Boolean)
<code>CByte(Expression)</code>	Chuyển đổi dữ liệu sang kiểu Byte
<code>CInt(Expression)</code>	Chuyển đổi dữ liệu sang kiểu nguyên (Integer)
<code>CLng(Expression)</code>	Chuyển đổi dữ liệu sang kiểu nguyên (Long)
<code>CDbl(Expression)</code>	Chuyển đổi dữ liệu sang kiểu thực (Double)
<code>CSng(Expression)</code>	Chuyển đổi dữ liệu sang kiểu thực (Single)
<code>CStr(Expression)</code>	Chuyển đổi dữ liệu sang kiểu chuỗi (String)
<code>Str(Number)</code>	Chuyển đổi dữ liệu số sang kiểu chuỗi (String)
<code>Val(String As String)</code>	Chuyển đổi dữ liệu từ String sang Double

Ví dụ:

```
Public Sub Test ()
    Dim StrA as String
    Dim A as Double
    StrA="1234"
    A=Val(StrA) ` Kết quả A=1234
    Debug.print A
    A=4567
    StrA=Str(A) ` Kết quả StrA="4567"
    Debug.Print StrA
End Sub
```

**GỢI Ý** Để có thể chạy thử các đoạn mã lệnh trên, trong VBA IDE, trước hết cần tạo ra một mô-đun trong dự án (nếu chưa có) sau đó tạo ra một chương trình con dạng `Sub` và nhập đoạn mã lệnh cần thử vào chương trình con này. Đặt con trỏ soạn thảo mã lệnh ở bất cứ dòng nào trong chương trình con đó và bấm phím `F5` để chạy chương trình.

Kết quả như sau:



**GỢI Ý** Cửa sổ Immediate là một bộ phận trong VBA IDE, bật / tắt cửa sổ này được thực hiện trong menu View của VBA IDE. Khi sử dụng lệnh `Debug.Print <tên_biến>` thì giá trị của biến sẽ được thể hiện trong cửa sổ Immediate khi chương trình hoạt động và được lưu lại ngay cả khi chương trình kết thúc. Cửa sổ này thường được dùng với mục đích gỡ rối khi lập trình. Khi nội dung trong cửa sổ này nhiều quá thì ta có thể xóa bớt bằng cách chọn vùng cần xóa và bấm phím `Delete`.

### 7.4. Các hàm xử lý chuỗi

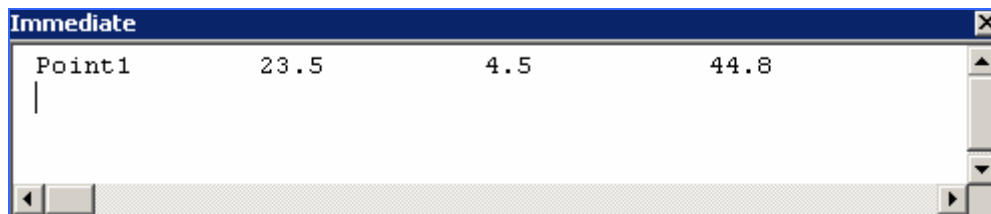
Các hàm loại này được chứa trong thư viện `Strings` (có thể tra cứu thư viện này bằng `Object Browser`). Sau đây là một số hàm thông dụng:

Hàm	Mô tả
<code>Asc(x)</code>	Trả về mã ASCII của ký tự đầu trong một chuỗi
<code>Chr(x)</code>	Chuyển đổi từ mã ASCII sang một ký tự
<code>Left(String, Length as Long)</code>	Trích dữ liệu bên trái của một chuỗi
<code>Mid(String, Start As Long, [Length])</code>	Trích dữ liệu phần giữa của một chuỗi
<code>Right(String, Length As Long)</code>	Trích dữ liệu phần bên phải của một chuỗi
<code>Split(String)</code>	Tách một chuỗi dài thành một mảng gồm nhiều chuỗi nhỏ hơn
<code>Joint(StringArray)</code>	Gộp một mảng các chuỗi thành một chuỗi duy nhất
<code>Len(String)</code>	Trả về độ dài của chuỗi (số lượng ký tự trong chuỗi bao gồm cả ký tự trống)
<code>Ucase(String)</code>	Hàm thực hiện đổi tất cả các ký tự trong chuỗi thành chữ HOA.
<code>InStr([start, ]string1, string2[, compare])</code>	Trả về vị trí bắt đầu của chuỗi <code>String2</code> trong chuỗi <code>String1</code> .

Ví dụ:

```
Public Sub Test()
    Dim StrArDes() As String
    ' Mảng các chuỗi được khai báo dạng mảng động
    Dim StrScr As String 'Chuỗi ban đầu
    StrScr = "Point1_23.5_4.5_44.8"
    StrArDes = Split(StrScr, "_")
    ' Tách chuỗi StrScr thành một mảng các chuỗi và đưa vào StrArDes,
    ' kí tự ngăn cách là "_"
    ' Khi đó StrArDes(0)="Point1", StrArDes(1)="23.5"
    ' StrArDes(2)="4.5", StrArDes(3)="44.8"
    Debug.Print StrArDes(0), StrArDes(1), StrArDes(2), StrArDes(3)
End Sub
```

Kết quả sẽ như sau:



Lưu ý là dấu “\_” trong ví dụ trên có thể thay thế bằng bất cứ ký tự nào.

**CHÚ Ý** Trong tất cả các ngôn ngữ lập trình, khái niệm chuỗi số và số là khác nhau. Ví dụ khi gán `A="123"` thì giá trị của `A` là một chuỗi ký tự gồm "1", "2" và "3". Còn khi gán `B=123` thì giá trị của `B` là **một trăm hai mươi ba**. Để tạo ra một chuỗi có chứa dấu nháy kép (") bên trong nó thì cần sử dụng thêm hai dấu nháy kép nữa. Ví dụ, trong biểu thức sau: `s = "ABC" "123"` thì giá trị của biến `s` là: `ABC"123`

## 8. Các cấu trúc điều khiển

### 8.1. Cấu trúc điều kiện

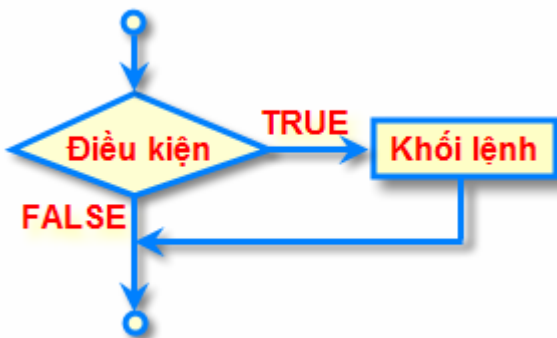
Các từ khóa: If, Then, Else, ElseIf, End If

Cú pháp:

```
If <biểu_thức_điều_kiện> then
    Khối_lệnh
End If
```

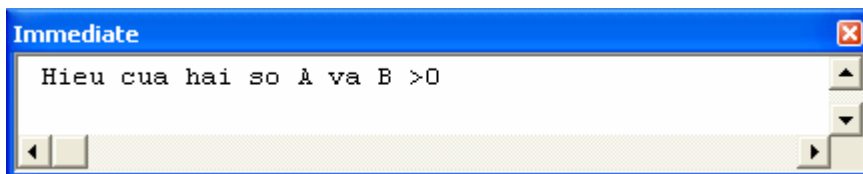
Diễn giải tiến trình của cấu trúc điều kiện như sau: nếu <biểu\_thức\_điều\_kiện> là đúng thì chương trình sẽ thực hiện <khối\_lệnh>, nếu sai thì chương trình sẽ thoát khỏi cấu trúc lệnh này.

Sơ đồ khối của cấu trúc lệnh kiểu này có thể được biểu diễn như sau:



```
Dim A As Double
Dim B As Double
A = 20: B = 10
If A > B Then Debug.Print ("Hieu cua hai so A va B >0")
```

Kết quả như sau:



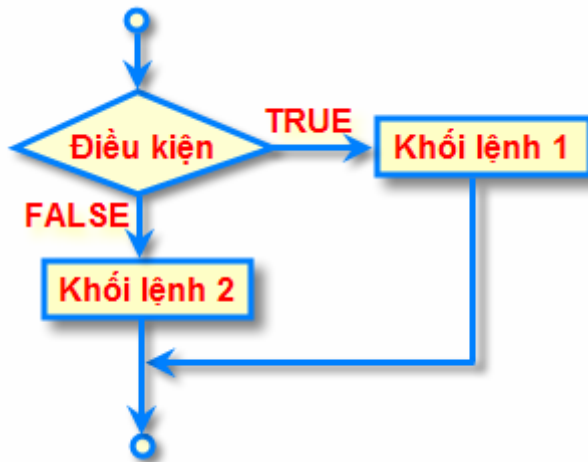
**GỢI Ý** Nếu như [khối\_lệnh] có thể viết trên một dòng như ví dụ trên thì không dùng từ khóa End If. Để phân tách nhiều lệnh trên cùng một dòng, sử dụng dấu hai chấm (:). Để ngăn cách giữa các lệnh.

Ngoài cấu trúc cơ bản và trường hợp riêng ở trên, trong nhiều trường hợp, ta buộc phải xử lý khi <Biểu\_thức\_điều\_kiện> trả về giá trị False (sai). Để giải quyết tình huống này ta sử dụng cấu trúc điều kiện mở rộng như sau:

```
If <biểu_thức_điều_kiện>
    Khối_lệnh_1
Else
    Khối_lệnh_2
End If
```

Diễn giải tiến trình của cấu trúc lệnh này như sau: nếu <biểu\_thức\_điều\_kiện> là đúng thì chương trình sẽ thực hiện <khối\_lệnh\_1>, còn nếu không đúng thì chương trình sẽ thực hiện <khối\_lệnh\_2>.

Sơ đồ khối của cấu trúc lệnh kiểu này có thể được biểu diễn như sau:



Các cấu trúc lệnh điều kiện có thể được lồng nhau để thể hiện những thao tác phức tạp hơn bằng cách sử dụng thêm từ khoá ElseIf. Như vậy, cấu trúc điều kiện có cú pháp tổng quát như sau:

```

If <điều_kiện_1> Then
  [Khối_lệnh_1]
  [ElseIf <điều_kiện_n> Then
    [khối_lệnh_n]
  ...
  [Else
    [Khối_lệnh_2]]
End If
  
```

Trong khối cấu trúc này, khối lệnh [ElseIf <điều\_kiện\_n> Then có thể lặp lại nhiều lần tương ứng với nhiều điều kiện khác nhau.

Diễn giải cấu trúc này như sau: nếu <điều\_kiện\_1> là đúng thì thực hiện [Khối\_lệnh\_1] và thoát khỏi khối cấu trúc này, còn nếu sai thì sẽ kiểm tra lần lượt từng điều kiện của ElseIf xem có giá trị nào đúng không, nếu không có giá trị nào đúng thì thực hiện [Khối\_lệnh\_2] (sau từ khóa Else) và thoát khỏi cấu trúc này, còn nếu gặp một giá trị đúng đầu tiên của <điều\_kiện\_n> nào đó thì khối lệnh tương ứng với ElseIf này sẽ được thực hiện và thoát khỏi cấu trúc này.

```

If (TheColorYouLike = vbRed) Then
  MsgBox "You 're a lucky person"
ElseIf (TheColorYouLike = vbGreen) Then
  MsgBox "You 're a hopeful person"
ElseIf (TheColorYouLike = vbBlue) Then
  MsgBox "You 're a brave person"
ElseIf (TheColorYouLike = vbMagenta) Then
  MsgBox "You 're a sad person"
Else
  MsgBox "You 're an average person"
End If
  
```

Ta xét ví dụ trên:

- ✦ Nếu `TheColorYouLike = vbRed` thì sẽ chỉ có thông báo: **You 're a lucky person.**
- ✦ Nếu `TheColorYouLike = vbBlue` thì sẽ chỉ có thông báo: **You 're a brave person.**
- ✦ Nếu `TheColorYouLike` không thuộc bất cứ giá trị nào trong bảng màu: `vbRed`, `vbGreen`, `vbBlue`, `vbMagenta` thì sẽ chỉ có thông báo: **You 're an average person.**

## 8.2. Cấu trúc lựa chọn

Cấu trúc này sử dụng khi ta muốn thực hiện một số lệnh nào đấy tương ứng với từng giá trị của biểu thức kiểm tra.

Các từ khoá sử dụng trong cấu trúc này: `Select Case`, `Case`, `Case Else`, `End Select`.

Cú pháp của cấu trúc lựa chọn:

```
Select Case <biểu_thức_kiểm_tra>
[Case điều_kiện_1
    [khối_lệnh_1]]
...
[Case điều_kiện_n
    [khối_lệnh_n]]
[Case Else
    [khối_lệnh_else]]
End Select
```

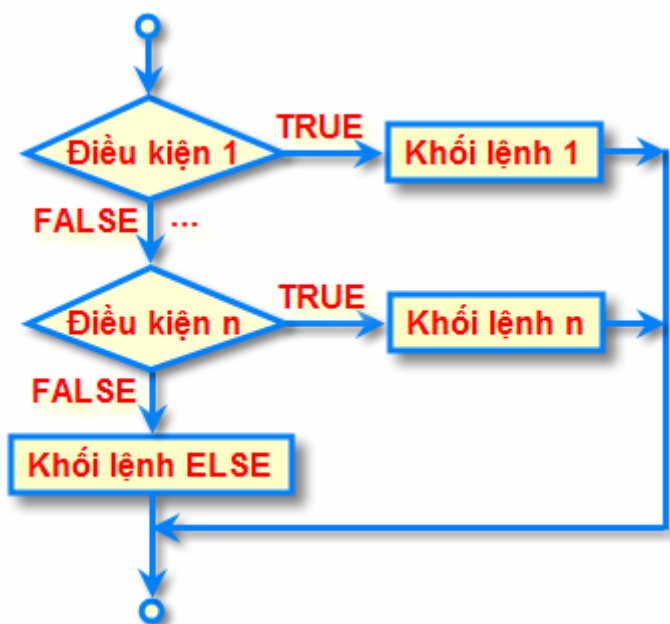
Diễn giải tiến trình của cấu trúc lựa chọn như sau: Giá trị của `<biểu_thức_kiểm_tra>` sẽ được so sánh với các `<điều_kiện_i>` nếu giá trị của `<biểu_thức_kiểm_tra>` thoả mãn `<điều_kiện_i>` thì `<khối_lệnh_i>` tương ứng sẽ được thực hiện, sau đó chương trình sẽ thoát khỏi cấu trúc lựa chọn. Trong trường hợp giá trị của `<biểu_thức_kiểm_tra>` không thoả mãn tất cả các điều kiện thì `<khối_lệnh_else>` sẽ được thực hiện nếu có từ khoá `Case Else`, còn nếu không có từ khoá `Case Else` thì chương trình sẽ thoát khỏi khối lệnh lựa chọn này mà không thực hiện gì cả.

Ví dụ sử dụng `ElseIf` ở trên được viết lại với cấu trúc lựa chọn như sau:

```
Select Case TheColorYouLike
Case vbRed
    MsgBox "You 're a lucky person"
Case vbGreen
    MsgBox "You 're a hopeful person"
Case vbBlue
    MsgBox "You 're a brave person"
Case vbMagenta
    MsgBox "You 're a sad person"
Else
    MsgBox "You 're an average person"
End Select
```

Có thể thấy rằng với cách viết sử dụng cấu trúc lựa chọn, đoạn chương trình trên dễ đọc hơn nhiều so với dùng cấu trúc điều kiện và `ElseIf`.

Sơ đồ khối của cấu trúc lựa chọn có thể được biểu diễn như sau:



### 8.3. Vòng lặp xác định

#### 8.3.1. Vòng lặp theo biến đếm

Thực hiện lặp một khối lệnh theo một biến đếm với số lần lặp xác định, ví dụ như khi ta cần tính tổng của các số nằm giữa hai số nào đó.

Các từ khóa: For, to, Step, Next

Cú pháp:

```
For <biến_đếm>=<Bắt_Đầu> To <Kết_Thúc> [Step <bước_nhảy>]
  [Khối_lệnh]
Next [<biến_đếm>]
```

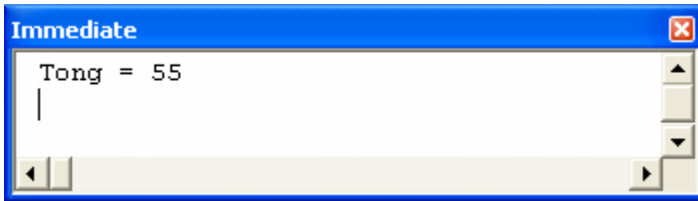
Cấu trúc lặp này thực hiện theo trình tự sau:

- ◆ Gán <Biến\_đếm> bằng giá trị <Bắt\_đầu>
- ◆ So sánh <Biến\_đếm> với giá trị <Kết\_thúc>:
  - Nếu *nhỏ hơn hoặc bằng*: thực hiện các lệnh bên trong [Khối\_lệnh] và tự động cộng vào <Biến\_đếm> một giá trị bằng <bước\_nhảy> nếu có từ khóa Step, còn không thì cộng thêm 1 và quay lại bước *so sánh* <Biến\_đếm> với giá trị <Kết\_thúc>.
  - Nếu *lớn hơn*: kết thúc khối lệnh lặp.

Ví dụ sau tính tổng của các số từ 1 đến 10:

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

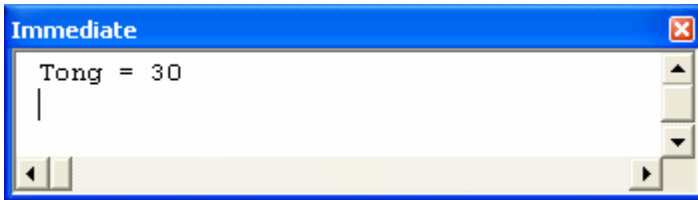
Kết quả như sau:



Ví dụ sau tính tổng của các số chẵn từ 0 đến 10:

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

Kết quả như sau:

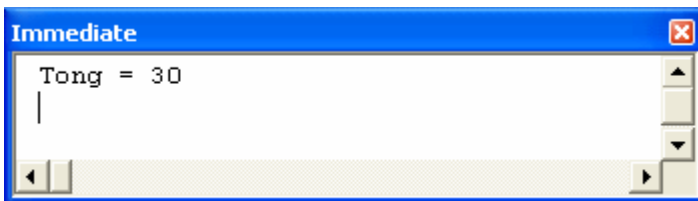


**CHÚ Ý** Khi giá trị của <bước\_nhảy> là âm (<0>) thì cấu trúc lặp sẽ thực hiện trình tự đếm ngược, nghĩa là vai trò của giá trị <bắt\_đầu> và <kết\_thức> đổi chỗ cho nhau.

Ví dụ tính tổng của các số chẵn từ 0 đến 10 sử dụng vòng lặp đếm ngược:

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

Kết quả như sau:



**GỢI Ý** Nếu như muốn thoát khỏi vòng lặp xác định FOR khi mà số lần lặp chưa đủ thì ta sử dụng từ khóa Exit For.

Ví dụ sau sẽ tính tổng của các số chẵn từ 0 đến 10, nhưng sẽ dừng vòng lặp FOR ngay khi tổng lớn hơn 20:

```
Dim i As Integer
Dim Tong As Integer
Tong = 0
For i = 10 To 0 Step -2
```



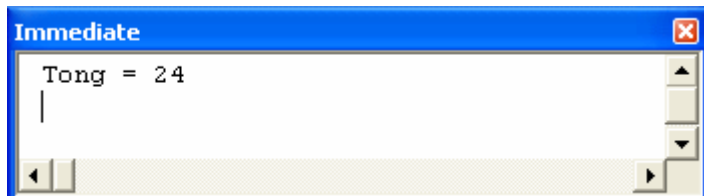
```
Tong = Tong + i
```

```
If Tong > 20 Then Exit For
```

```
Next
```

```
Debug.Print ("Tong = " & Tong)
```

Kết quả như sau: (10 + 8 + 6 = 24)



### 8.3.2. Lặp trong một tập hợp

Trong trường hợp muốn thực hiện các khối lệnh lặp theo một biến đếm chạy trong một tập hợp mà tập hợp đó không thể xác định được số lượng hoặc bước nhảy thì người dùng có thể dùng vòng lặp trong tập hợp (For Each ... Next). Tập hợp ở đây có thể là một tập đối tượng dạng Collection hoặc một mảng.

Các từ khoá sử dụng For, Each, In, Next

Cú pháp:

```
For Each <biến_chạy> In <tập_hợp>
    [Khối_lệnh]
Next
```

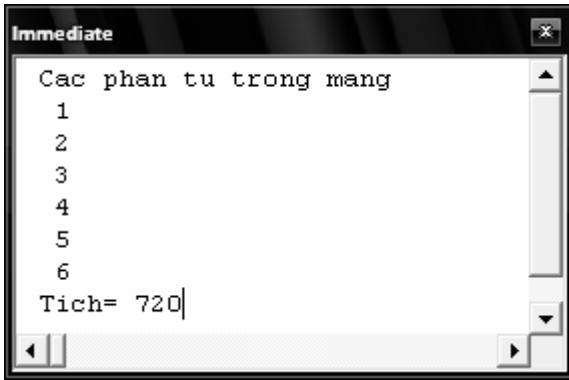
Giải thích: <biến\_chạy> sẽ nhận các giá trị từ phần tử đầu tiên đến phần tử cuối cùng trong <tập\_hợp>. Ứng với mỗi giá trị của <biến\_chạy>, khối lệnh được thực hiện một lần.

**CHÚ Ý** Kiểu của <biến\_chạy> trong vòng lặp (For Each ... Next) sẽ phụ thuộc vào kiểu của <tập\_hợp> mà nó duyệt qua là kiểu mảng hay kiểu tập đối tượng. Đối với <tập\_hợp> là tập đối tượng thì kiểu dữ liệu của <biến\_chạy> có thể là Variant, hoặc đối tượng cùng kiểu với tập đối tượng đó. Đối với <tập\_hợp> là mảng thì kiểu dữ liệu của <biến\_chạy> chỉ có thể là Variant.

Ví dụ sau sẽ thực hiện tính tích các số trong một mảng 2 chiều với việc dùng vòng lặp trong tập hợp. Kết quả sẽ được hiển thị trong cửa sổ Immediate.

```
Public Sub TestForEach()
    Dim a(0 To 2, 0 To 1) As Double
    Dim v As Variant
    Dim Tich As Double
    a(0, 0) = 1: a(1, 0) = 2: a(2, 0) = 3
    a(0, 1) = 4: a(1, 1) = 5: a(2, 1) = 6
    Tich = 1
    Debug.Print "Cac phan tu trong mang"
    For Each v In a
        Debug.Print v
        Tich = Tich * v
    Next
    Debug.Print "Tich=" & Str(Tich)
End Sub
```

Kết quả như sau:



## 8.4. Vòng 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). Tùy thuộc vào việc kiểm tra *biểu thức điều kiện* mà ta sử dụng một trong hai dạng cú pháp như sau:

### Kiểu 1: Lặp trong khi biểu thức điều kiện là TRUE

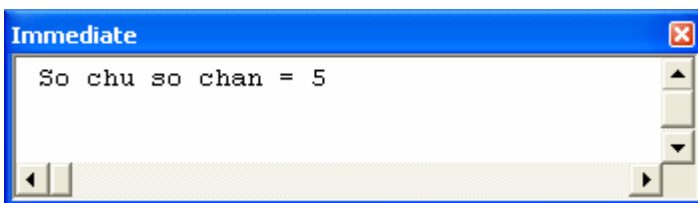
```
Do While <điều_kiện>
    [Khối_lệnh]
Loop
```

Với cú pháp này, [Khối\_lệnh] chỉ được thực hiện khi <Điều\_kiện> là đúng.

Ví dụ sau sẽ đếm số chữ số chẵn trong khoảng hai số A, B:

```
Dim i, A, B, SoChan As Integer
A = 1: B = 10
i = A
SoChan = 0
Do While i <= B
    If (i Mod 2) = 0 Then SoChan = SoChan + 1
    i = i + 1
Loop
Debug.Print ("So chu so chan = " & SoChan)
```

Kết quả như sau:



Nếu muốn vòng lặp luôn có ít nhất một lần thi hành khối lệnh, sử dụng cú pháp:

```
Do
    [Khối_lệnh]
Loop While <điều_kiện>
```

Với cú pháp này, [Khối\_lệnh] được thực hiện ít nhất một lần cho dù <Điều\_kiện> đúng hay sai bởi <Điều\_kiện> được kiểm tra ở cuối của cấu trúc.

### Kiểu 2: Lặp cho đến khi điều kiện là FALSE

```
Do Until <điều_kiện>
    [Khối_lệnh]
Loop
```

Nếu muốn vòng lặp luôn có ít nhất một lần thì hành khối lệnh sử dụng cú pháp:

```
Do
    [Khối_lệnh]
Loop Until <điều_kiện>
```

**CHÚ Ý** Khi [Khối\_lệnh] được thực thi, nếu như trong [Khối\_lệnh] không có câu lệnh nào tác động lên <điều\_kiện> để nó nhận giá trị ngược lại thì vòng lặp này sẽ không bao giờ kết thúc và làm cho ứng dụng bị "treo". Để thoát khỏi tình huống "treo" này có nhiều cách và cách đơn giản nhất là bấm tổ hợp phím Ctrl+Break để quay trở lại VBAIDE.

Có cách khác để thoát khỏi vòng lặp, ngoài việc thiết lập <điều\_kiện> có giá trị ngược lại, là sử dụng từ khóa `Exit Do` đặt trong [Khối\_lệnh].

## 9. Chương trình con

Về cơ bản, chương trình con là một khối các câu lệnh và chúng được sử dụng lặp lại trong chương trình chính thông qua tên của chương trình con. Chương trình con đặc biệt hữu ích khi thay thế các khối lệnh lặp nhau hoặc cùng thực thi một chức năng tương tự nào đó.

Có hai loại chương trình con chính là Hàm (Function) và Thủ tục (Sub). Ngoài ra, trong các mô-đun lớp (Class Module) còn có chương trình con dạng thuộc tính (Property), tuy nhiên trong giáo trình này sẽ không trình bày về loại chương trình con này mà người đọc có thể tham khảo trong giáo trình môn *Lập trình hướng đối tượng trong xây dựng*.

Cú pháp tổng quát của một chương trình con như sau:

Cú pháp tổng quát của một chương trình con như sau:

```
[Private|Friend|Public] [Static] <Sub|Function|Property>
Tên ([các_tham_số])
    [Khối_lệnh]
End <Sub|Function|Property>
```

Trong đó phần thân chương trình con được bọc giữa phần khai báo và phần kết thúc (có từ khóa `End`).

Các từ khóa [Private|Public|Friend] xác định phạm vi hoạt động của chương trình con. Khái niệm phạm vi này cũng tương tự như phạm vi của biến đã được trình bày ở phần trước.

Từ khóa [Static] xác định cách thức cấp phát bộ nhớ cho các biến khai báo bên trong chương trình con (sẽ trình bày cụ thể ở phần sau).

**CHÚ Ý** Từ khóa `Friend` chỉ được sử dụng trong mô-đun lớp hoặc mô-đun lệnh của UserForm.

## 9.1. Hàm (Function)

Là chương trình con có trả về giá trị khi nó được gọi. Cú pháp khai báo như sau:

```
[Private/Public/Friend] [Static] Function <Tên_hàm> ([Các_tham_số]) as  
<kiểu_dữ_liệu>  
    [Khởi_lệnh]  
End Function
```

Ví dụ: tạo hàm tính diện tích của hình chữ nhật, với hai tham số cần nhập vào là chiều rộng và chiều dài của 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)

Là chương trình con không trả về giá trị khi được gọi. Cú pháp khai báo như sau:

```
[Private/Public/Friend] [Static] Sub <Tên_hàm> ([Các_tham_số])  
    [Khởi_lệnh]  
End Sub
```

Ví dụ: để tạo một chương trình con dạng thủ tục có tính năng như phần trên có thể viết mã lệnh như sau:

```
Sub Dien_Tich(Rong as Double, Dai as Double, Dt as Double)  
    Dt=Rong*Dai  
End Sub
```

**CHÚ Ý** Trong ví dụ này, vì chương trình con không có giá trị trả về nên để nhận về giá trị diện tích phải bổ sung thêm tham số Dt vào trong danh sách tham số của chương trình con.

## 9.3. Truyền tham số cho chương trình con

Xét 2 chương trình con được đặt trong cùng một mô-đun chuẩn, thực hiện việc gán và in giá trị của biến như sau:

1 Một chương trình con đơn giản được tạo ra như sau:

```
Public Sub Test(ByRef a As Long, b As Long, ByVal c As Long)  
    a = 100: b = 200: c = 300  
End Sub
```

Chú ý đến khai báo biến a, b và c của chương trình con này:

- ✦ Trước biến a là từ khóa ByRef.
- ✦ Trước biến b không có từ khóa, nghĩa là sử dụng kiểu mặc định của VB.
- ✦ Trước biến c là từ khóa ByVal.

2 Chương trình con thứ hai được xây dựng trên cùng một mô-đun với chương trình con trên như sau:

```
Public Sub CallTest()
    Dim va As Long, vb As Long, vc As Long
    va = 500: vb = 500: vc = 500
    ' In giá trị của biến trước khi gọi chương trình con thứ nhất
    Debug.Print " Cac gia tri bien truooc khi goi chuong trinh con:"
    Debug.Print "va=" & Str(va)
    Debug.Print "vb=" & Str(vb)
    Debug.Print "vc=" & Str(vc)
    ' Gọi chương trình con thứ nhất
    Test va, vb, vc
    ' In giá trị của biến sau khi gọi chương trình con thứ nhất
    Debug.Print " Cac gia tri bien sau khi goi chuong trinh con:"
    Debug.Print "va=" & Str(va)
    Debug.Print "vb=" & Str(vb)
    Debug.Print "vc=" & Str(vc)
End Sub
```

Trong chương trình con thứ 2 có lời gọi đến chương trình con thứ nhất để thực hiện thay đổi giá trị của các biến. Kết quả khi thực thi chương trình con thứ 2 như sau:

Immediate

```
Cac gia tri bien truooc khi goi chuong trinh con:
va= 500
vb= 500
vc= 500
Cac gia tri bien sau khi goi chuong trinh con:
va= 100
vb= 200
vc= 500
```

→ Các biến tham chiếu, giá trị bị thay đổi

→ Biến tham trị, giá trị không bị thay đổi

Qua kết quả trên có thể thấy rằng:

Giá trị của biến *có thể bị thay đổi hoặc không bị thay đổi* khi chúng được truyền vào chương trình con là phụ thuộc vào cách *định nghĩa tham số trong chương trình con* đó.

- ◆ Biến a trong Sub Test được khai báo với từ khóa ByRef và khi truyền biến ở vị trí này (biến va trong CallTest) thì giá trị của biến ban đầu bị thay đổi tương ứng với các tác động trong chương trình con.
- ◆ Biến b trong Sub Test được khai báo mặc định (không có từ khóa nào phía trước nó) và khi truyền biến ở vị trí này (biến vb trong CallTest) thì giá trị của biến ban đầu bị thay đổi tương ứng với các tác động trong chương trình con.
- ◆ Biến c trong Sub Test được khai báo với từ khóa ByVal và khi truyền biến ở vị trí này (biến vc trong CallTest) thì giá trị của biến ban đầu **không** bị thay đổi cho dù trong chương trình con biến này bị tác động.

Qua ví dụ trên có thể thấy rằng việc truyền tham số cho chương trình con có thể được phân làm hai trường hợp và được đặt tên là *truyền tham số theo tham chiếu* và *truyền tham số theo tham trị*.

### 9.3.1. Truyền tham số theo tham chiếu

Khi truyền một biến vào tham số theo kiểu tham chiếu, địa chỉ của biến sẽ được truyền cho chương trình con. Do đó, bất kì câu lệnh nào của chương trình con tác động lên tham số sẽ ảnh hưởng trực tiếp lên biến được truyền tương ứng, nghĩa là khi chương trình con kết thúc, giá trị của biến được truyền theo kiểu này sẽ bị thay đổi do chương trình con. Truyền tham số theo

kiểu tham chiếu là mặc định trong VB, người dùng cũng có thể chỉ rõ việc truyền theo tham chiếu bằng cách thêm từ khoá `ByRef` vào trước khai báo tham số.

### 9.3.2. Truyền tham số theo tham trị

Khi truyền một biến vào tham số theo kiểu tham trị, bản sao giá trị của biến sẽ được truyền cho cho chương trình con. Do đó, nếu trong chương trình con có các câu lệnh tác động lên tham số thì chỉ bản sao bị ảnh hưởng và biến truyền vào sẽ không bị thay đổi, nghĩa là sau khi chương trình con kết thúc, giá trị của biến vẫn được giữ nguyên như ban đầu. Để xác định cách thức truyền dữ liệu cho một tham số theo kiểu tham trị, thêm từ khoá `ByVal` vào trước khai báo tham số.

Trong `Sub Test` ở trên, `a` và `b` là hai tham số được truyền theo kiểu tham chiếu còn `c` được truyền theo kiểu tham trị.

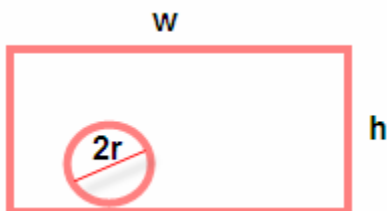
### 9.3.3. Tham số tùy chọn.

Tham số tùy chọn là tham số có thể có hoặc được bỏ qua khi gọi chương trình con.

Các tham số tùy chọn được khai báo với từ khoá **Optional** và trong một chương trình con, *các khai báo của các tham số tùy chọn luôn phải nằm cuối danh sách tham số được khai báo.*

Ví dụ: viết chương trình con tính toán diện tích của mặt cắt chữ nhật có khoét lỗ (như hình dưới) với yêu cầu sau:

- ◆ Tính diện tích mặt cắt với các thông số về chiều rộng `w`, chiều cao `h` và bán kính `r` của lỗ khoét.
- ◆ Trong trường hợp thiếu thông số về bán kính `r`, chỉ tính diện tích mặt cắt chữ nhật và bỏ qua lỗ khoét.



Dưới đây là một chương trình con có sử dụng tham số tùy chọn:

```
Public Function DT(w As Double, h As Double, Optional r As Variant)
    If Not IsMissing(r) Then
        If (2 * r <= w) And (2 * r <= h) Then
            DT = w * h - pi * r ^ 2
        Else
            MsgBox " Co loi, lo khoet vuot ra ngoai hinh"
            DT = "Error"
        End If
    Else
        DT = w * h
    End If
End Function
```

Sau khi tạo mã lệnh trên, nếu muốn tính diện tích cho mặt cắt với `w = 100`, `h = 200`, `r = 20` có thể gọi hàm như sau: `DT(100, 200, 20)` để tính diện tích có xét đến khoét lỗ với bán kính là 20, hoặc `DT(100, 200)` để tính diện của hình chữ nhật (không có lỗ).

**CHÚ Ý** Để biết được một tham số tùy chọn có bị bỏ qua khi gọi chương trình con hay không, dùng hàm `IsMissing(tham_số_tùy_chọn)` và đồng thời tham số tùy chọn bắt buộc phải có kiểu dữ liệu là `Variant` (vì hàm `IsMissing` chỉ có hiệu lực đối với các biến kiểu `Variant`). Hàm này trả về `TRUE` nếu tham số bị bỏ qua, `FALSE` nếu tham số có mặt.

### 9.3.4. Danh sách tham số với số lượng tham số tùy ý.

Visual Basic 6.0 cho phép tạo một chương trình con với danh sách tham số tùy ý (nghĩa là số lượng các tham số có thể thay đổi khi gọi chương trình con) thông qua việc đặt từ khóa `ParamArray` trước danh sách tham số. Khi đó danh sách tham số là tùy chọn và có dạng một mảng kiểu `Variant`.

Ví dụ: viết một hàm tính tổng của tất cả các số truyền vào với số lượng số được truyền là tùy ý.

Mã lệnh tham khảo như sau:

```
Public Function TinhTong(ParamArray ds())
    Dim So As Variant
    Dim Tong As Variant
    Tong = 0
    For Each So In ds
        Tong = Tong + So
    Next
    TinhTong = Tong
End Function
```

Khi đó:

`TinhTong(100, 200, -200)` cho kết quả là 100

`TinhTong(2, 300)` cho kết quả là 302

### 9.3.5. Hàm có giá trị trả về là kiểu mảng.

Để khai báo một hàm trả về mảng, thêm cặp kí tự “( )” sau khai báo hàm

```
[Private/Public] Function <Tên_hàm> ([danh sách tham số]) as _
<kiểu_dữ_liệu> ()
    [Khởi_lệnh]
End Function
```

Ví dụ: viết chương trình con sắp xếp các phần tử trong mảng một chiều và trả về một mảng có thứ tự tăng dần.

Mã lệnh tham khảo như sau:

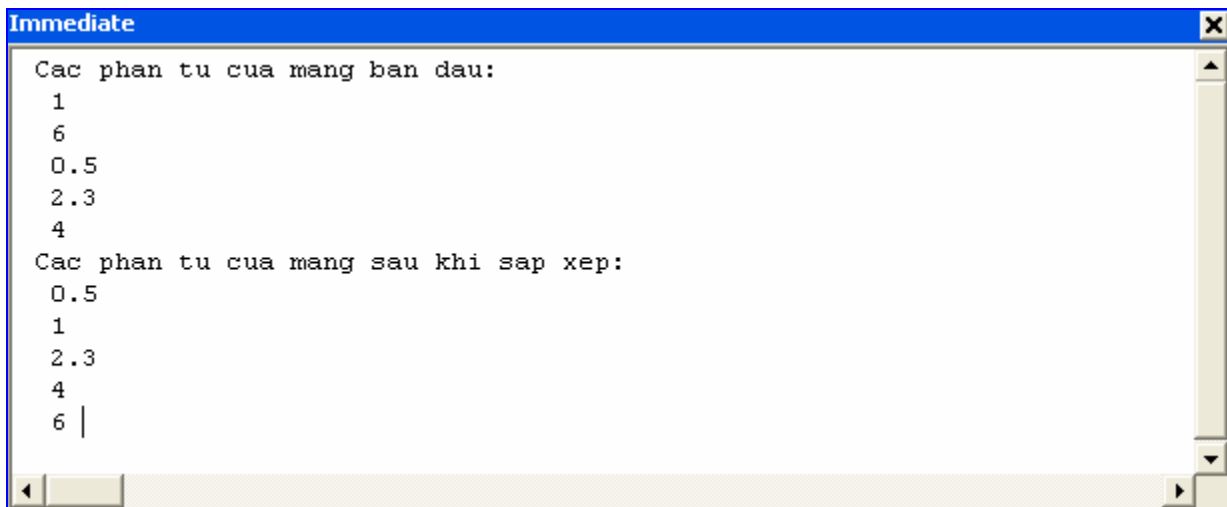
```
Public Function Mang_tangdan(Mang_bandau() As Double) As Double()
    Dim Lb As Long, Ub As Long ' bien dau va cuoi cua mang
    Dim i As Long, j As Long
    Lb = LBound(Mang_bandau) : Ub = UBound(Mang_bandau)
    Dim Mang_tamthoi() As Double ' Khai bao mot mang tam thoi
    Mang_tamthoi = Mang_bandau
    Dim Tg As Double
    For i = Lb To Ub - 1
        For j = i + 1 To Ub
            If Mang_tamthoi(i) > Mang_tamthoi(j) Then
                Tg = Mang_tamthoi(i)
                Mang_tamthoi(i) = Mang_tamthoi(j)
```

```
        Mang_tamthoi(j) = Tg
    End If
Next
Next
Mang_tangdan = Mang_tamthoi
Erase Mang_tamthoi ' Huy mang tam thoi
End Function
```

Chương trình thử nghiệm hàm trên:

```
Public Sub test()
    Dim a(2 To 6) As Double
    a(2) = 1: a(3) = 6: a(4) = 0.5: a(5) = 2.3: a(6) = 4
    Dim b() As Double
    b = Mang_tangdan(a) ' Goi ham da viet
    Dim so As Variant
    Debug.Print "Cac phan tu cua mang ban dau:"
    For Each so In a
        Debug.Print so
    Next
    Debug.Print "Cac phan tu cua mang sau khi sap xep:"
    For Each so In b
        Debug.Print so
    Next
End Sub
```

Kết quả như sau:



## 9.4. Biến trong chương trình con

Như đã trình bày ở phần trước, biến trong chương trình con luôn có tính chất cục bộ. Tuy nhiên hình thức cấp phát bộ nhớ cho biến thì có thể khác nhau. Tùy vào từng trường hợp cụ thể:

① Trường hợp: trong phần khai báo của chương trình con không sử dụng từ khóa Static

Với các biến được khai báo bình thường với từ khóa Dim: mỗi lần chương trình con được gọi, biến sẽ được tạo và cấp phát bộ nhớ. Khi chương trình con kết thúc, bộ nhớ dành cho biến được giải phóng. Do đó, giá trị của biến sau mỗi phiên làm việc của chương trình con sẽ không được lưu trữ.

Với các biến được khai báo với từ khóa Static: biến sẽ được khởi tạo một lần khi mô-đun chứa chương trình con được nạp vào trong bộ nhớ và sẽ tồn tại trong bộ nhớ cùng với mô-đun đó. Vì

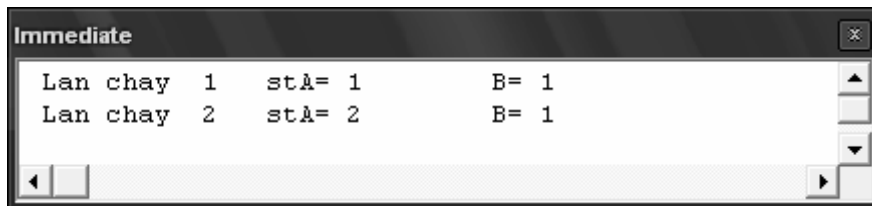


vậy, giá trị của biến sau mỗi phiên làm việc của chương trình con sẽ được lưu trữ. Các biến kiểu này được gọi là biến tĩnh (Static)

Ví dụ: trong chương trình con StVariable dưới đây có hai biến địa phương, stA là biến tĩnh và B là biến thông thường.

```
Public Sub StVariable()
    Static stA As Long
    Dim B As Long
    B = B + 1
    stA = stA + 1
    Debug.Print "Lan chay " & Str(stA), "stA=" & Str(stA), "B=" &
    Str(B)
End Sub
```

Kết quả sau 2 lần chạy chương trình con trên như sau:



### Giải thích

Ngay khi được khai báo, tất cả các biến đều được tự động khởi tạo giá trị ban đầu, nếu kiểu dữ liệu của biến là dạng số thì giá trị khởi tạo bằng 0, còn nếu kiểu dữ liệu của biến là chuỗi thì giá trị khởi tạo mặc định là chuỗi rỗng. Trong chương trình trên, ngay trước khi kết thúc ở lần chạy đầu tiên, giá trị của các biến như sau:

- ◆ Biến B = 1.
- ◆ Biến stA = 1.

Khi kết thúc lần chạy thứ nhất, biến B (biến thông thường) sẽ được giải phóng, còn biến stA (biến tĩnh) vẫn được lưu giá trị (=1) của nó lại trong bộ nhớ. Do đó đến lần chạy thứ hai, biến B được tạo mới sẽ nhận giá trị là  $B=B+1=0+1=1$ , còn biến stA do vẫn tồn tại từ lần trước nên giá trị của nó là  $stA=stA+1=1+1=2$ .

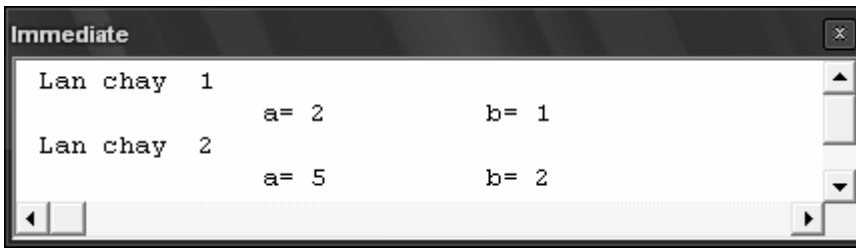
**2** Trường hợp: trong khai báo của chương trình con có sử dụng từ khóa Static

Khi đó tất cả các biến khai báo trong chương trình con sẽ là các biến tĩnh.

Ví dụ: trong chương trình con StPro dưới đây đã sử dụng khai báo Static ở đầu chương trình.

```
Public Static Sub StPro()
    Dim a As Long
    Dim b As Long
    a = a + 1
    b = b + 1
    a = a + b
    Debug.Print "Lan chay " & Str(b)
    Debug.Print "    ", "a=" & Str(a), "b=" & Str(b)
End Sub
```

Kết quả sau 2 lần chạy chương trình con như sau:



**CHÚ Ý** Các biến tĩnh thường được sử dụng khi muốn lưu trữ kết quả những lần chạy của chương trình con. Chú ý rằng dù biến trong chương trình con là biến thông thường hay biến tĩnh thì vẫn luôn mang tính chất cục bộ.

### 9.5. Cách thức gọi chương trình con.

Với trường hợp dự án (Project) gồm nhiều thành phần (các mô-đun chuẩn, các UserForm,...) có chứa mã lệnh, nghĩa là ở đó có thể xây dựng hoặc có nhu cầu sử dụng chương trình con, thì trong cùng một mô-đun, không được phép xây dựng hai chương trình con trùng tên nhau, nhưng quy định này không áp dụng cho các mô-đun khác nhau, nghĩa là có thể tồn tại hai chương trình con có tên giống hệt nhau ở hai mô-đun khác nhau. Trong trường hợp trùng tên này, khi muốn sử dụng chương trình con nào thì phải chỉ rõ nơi chứa nó, và tốt nhất, khi sử dụng bất cứ chương trình con nào của mô-đun khác thì nên chỉ rõ cả tên mô-đun đó.

#### Gọi chương trình con dạng hàm (Function)

Khi gọi chương trình con dạng hàm (Function), *danh sách tham số phải được đặt trong cặp kí tự “( )”* sau tên chương trình con.

**<Tên\_mô-đun>.<Tên\_hàm>(<danh\_sách\_tham\_số>)**

**CHÚ Ý** Mô-đun ở đây có thể là một mô-đun chuẩn (Module), UserForm hoặc một đối tượng mà người dùng đang xét. Danh sách tham số phải được truyền theo đúng thứ tự như ở phần khai báo chương trình con.

Ví dụ: mô-đun chuẩn mdlMatcat chứa hàm TinhDTHH(h,b) thì cú pháp gọi hàm đó là:

mdlMatcat.TinhDTHH(ph,pb)

với ph, pb là những biến được truyền vào trong hàm.

#### Gọi chương trình con dạng thủ tục (Sub)

Khi gọi chương trình con dạng thủ tục (Sub), danh sách tham số đặt tiếp sau tên thủ tục và kí tự trống, *các tham số không cần đặt trong cặp kí tự “( )”*.

**<Tên\_mô-đun>.<Tên\_thủ\_tục> <danh\_sách\_tham\_số>**

Ví dụ: trong mô-đun chuẩn mdlDAH chứa thủ tục TinhDTDAH(S) thì cú pháp gọi thủ tục đó là:

mdlDAH.TinhDTDAH pS

với pS là những biến được truyền vào trong thủ tục.

#### Gọi chương trình con với các tham số gán theo tên

Trong cách gọi chương trình con theo kiểu thông thường như trên, danh sách tham số truyền vào phải đúng thứ tự như trong phần khai báo của chương trình con đó. Ngoài ra, VB còn cho

phép gọi chương trình con với trật tự tham số tùy ý mà vẫn đảm bảo sự truyền tham số chính xác thông qua tên của tham số.

Ví dụ, với hàm DT(w, h, r) ở phần trên thì hai cách gọi sau là tương đương:

```
DT (100, 200, 30)
DT (r:=30, w:=100, h:=200)
```

Trong dòng thứ nhất, luôn có sự ngầm hiểu trình tự các tham số là w, h, r, đây chính là trình tự khi định nghĩa hàm DT. Còn ở dòng thứ 2, trình tự theo định nghĩa của hàm DT không có ý nghĩa nữa bởi đã có sự chỉ rõ: Tên biến := Giá trị cần gán. Chú ý đến ký hiệu (:=) và trình tự bất kỳ của các tham số.

Việc sử dụng tham số gán theo tên khi gọi chương trình con đặc biệt tiện lợi khi chương trình con có nhiều tham số tùy chọn và người dùng không có ý định sử dụng hết các tham số đó.

### 9.6. Thoát khỏi chương trình con.

Để thoát khỏi hàm sử dụng lệnh Exit Function

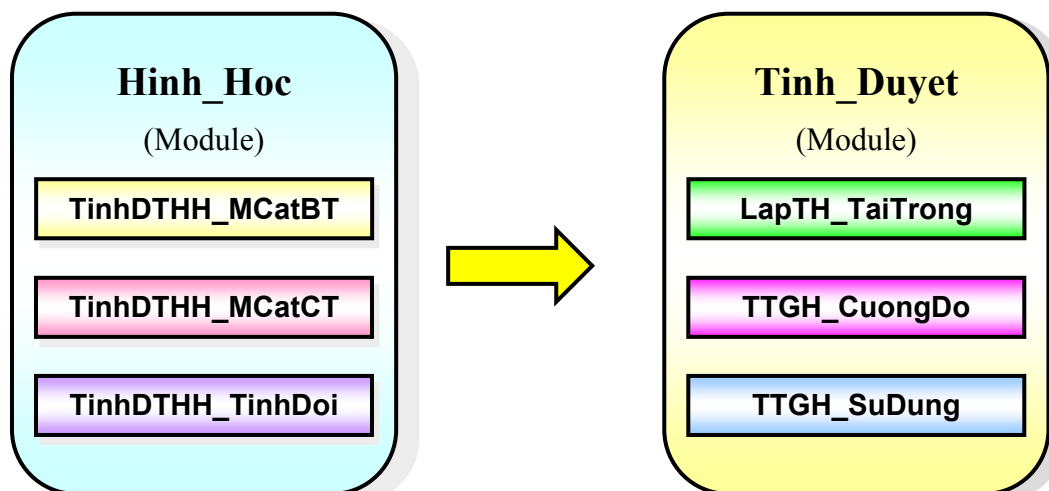
Để thoát khỏi thủ tục sử dụng lệnh Exit Sub

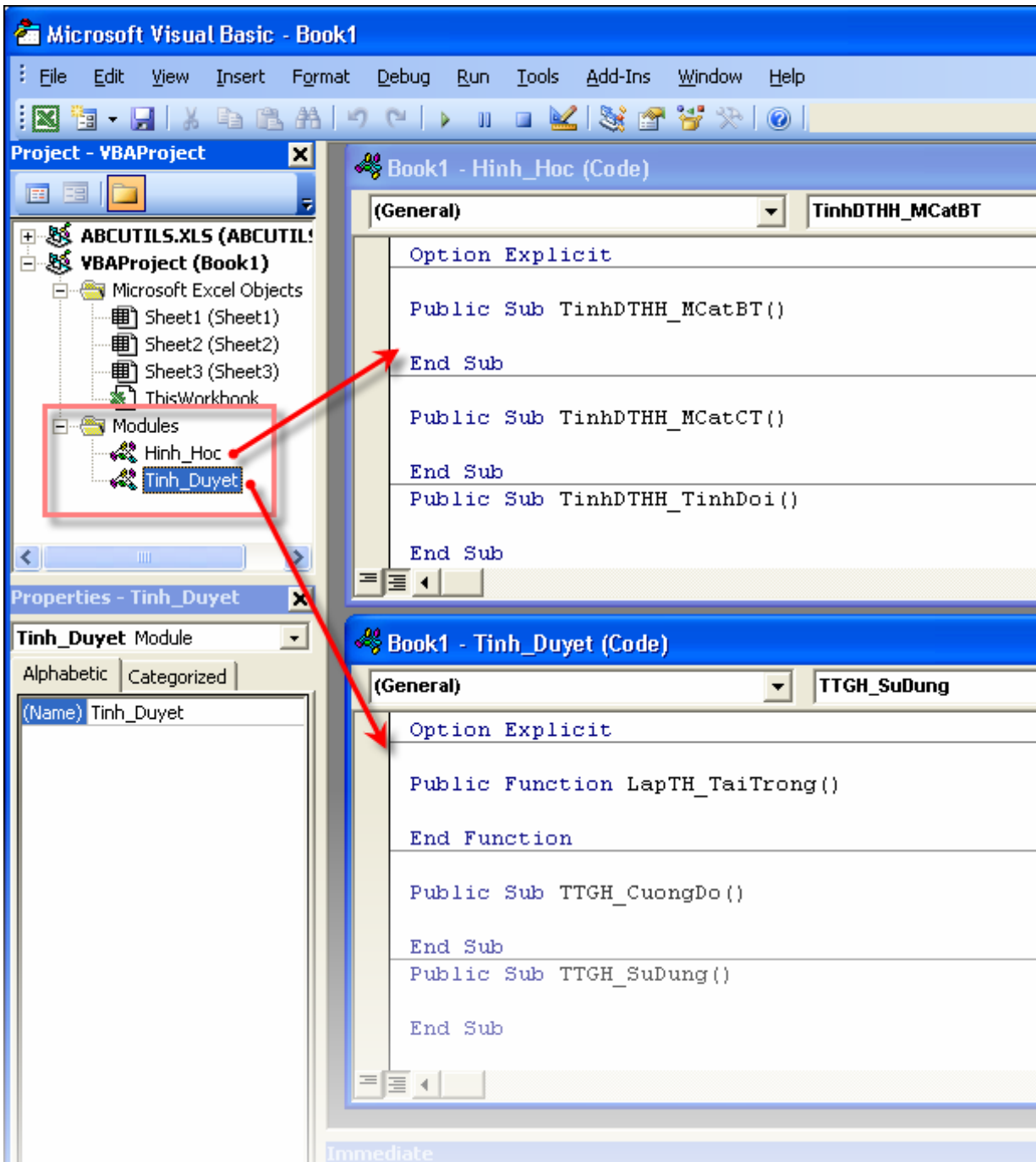
Ngay khi gặp hai hàm này trong thân của chương trình con, toàn bộ các dòng lệnh phía sau nó sẽ bị bỏ qua và chương trình sẽ thoát ngay khỏi chương trình con đó.

## 10. Tổ chức các chương trình con theo hệ thống các mô-đun chuẩn

Với việc thiết kế hệ thống theo phương pháp cấu trúc hóa, toàn bộ chương trình thường được chia thành các khối chương trình nhỏ hơn, mỗi khối chương trình đảm nhận một chức năng chung nào đó. Tiếp theo, để dễ dàng cho việc xây dựng chương trình, các chức năng chung lại được chia thành các phần nhỏ hơn nữa, và lặp lại cho đến khi nào mỗi phần này có thể minh họa bằng một chương trình con. Trong lập trình VBA, các khối chức năng thường được tổ chức thành các mô-đun chuẩn (Module). Trong mô-đun chuẩn sẽ bao gồm các chương trình con (hàm và thủ tục) phản ánh sự chi tiết hoá cho các khối chức năng này. Ngoài ra, trong mô-đun chuẩn người dùng có thể khai báo các kiểu dữ liệu tự định nghĩa, các biến dùng chung, các hằng số, ...

Ví dụ: để xây dựng một chương trình kiểm toán mặt cắt cột BTCT, có thể xây dựng các mô-đun và các chương trình con theo hình vẽ dưới đây dưới đây





Hình III-16: Tổ chức dự án theo cấu trúc chức năng

## 11. Làm việc với UserForm và các thành phần điều khiển

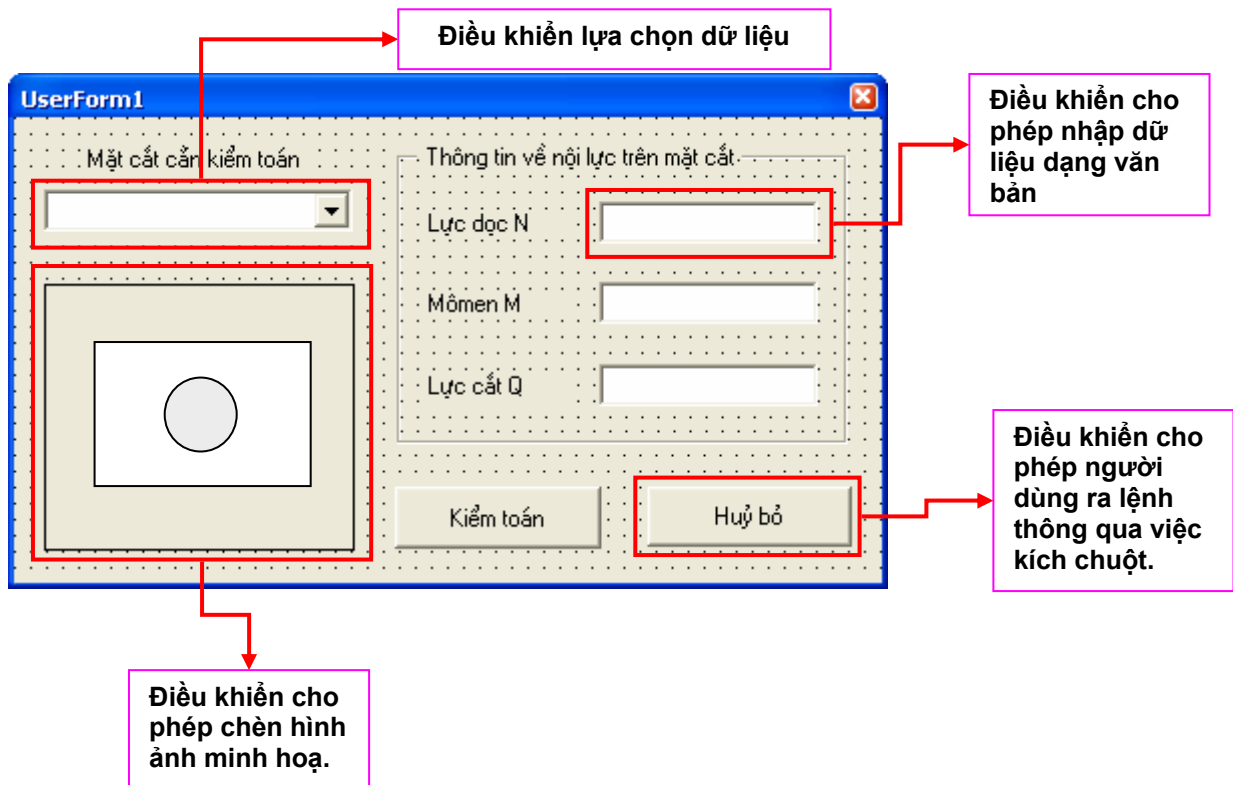
### 11.1. Các vấn đề chung

Trong một dự án VBA, các mô-đun chuẩn cho phép xây dựng các khối chương trình xử lý dữ liệu hoặc các khai báo về dữ liệu. Sự giao tiếp nhập-xuất dữ liệu giữa người dùng và chương trình có thể được thực hiện thông qua giao diện của ứng dụng nên. Tuy nhiên, trong nhiều trường hợp giao diện nhập-xuất dữ liệu của ứng dụng nên chưa thể đáp ứng được nhu cầu tương tác dữ liệu một cách chi tiết cũng như tiện lợi cho người sử dụng, và khi đó, cần tạo ra các giao diện nhập-xuất riêng thông qua việc sử dụng các UserForm trong dự án VBA. Nói cách khác, giao tiếp giữa người sử dụng chương trình với chương trình viết bằng VBA được gọi là giao diện của chương trình và cách xây dựng giao diện như sau:

- ◆ Sử dụng ngay ứng dụng nền để làm giao diện, cách này sẽ trình bày cụ thể trong các chương sau.

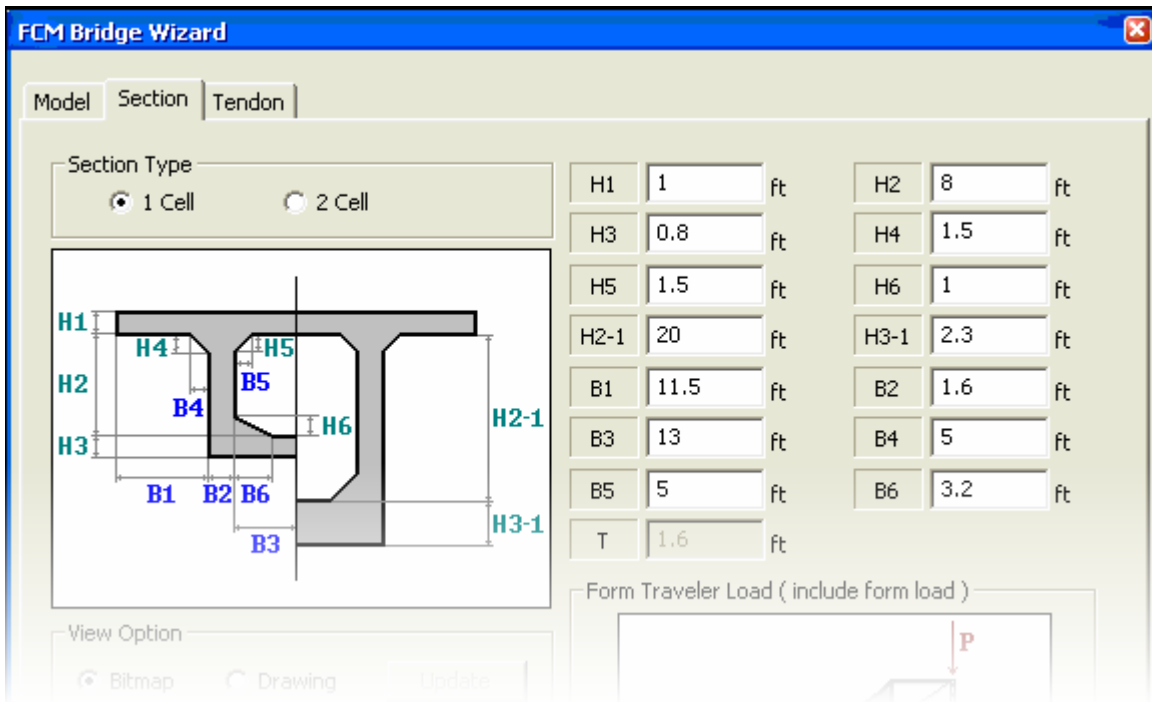
- ◆ Sử dụng UserForm.
- ◆ Kết hợp cả hai phương án trên.

Các UserForm thực chất là mẫu các hộp thoại (cửa sổ) được tạo ra theo yêu cầu của người dùng. Trên một UserForm luôn chứa những thành phần phục vụ cho nhu cầu tương tác giữa người dùng và chương trình: nhập các dữ liệu cần thiết, ra lệnh xử lý, lựa chọn dữ liệu theo tình huống, hiển thị kết quả xử lý một cách trực quan,... Những thành phần đó được gọi là các điều khiển (Control).



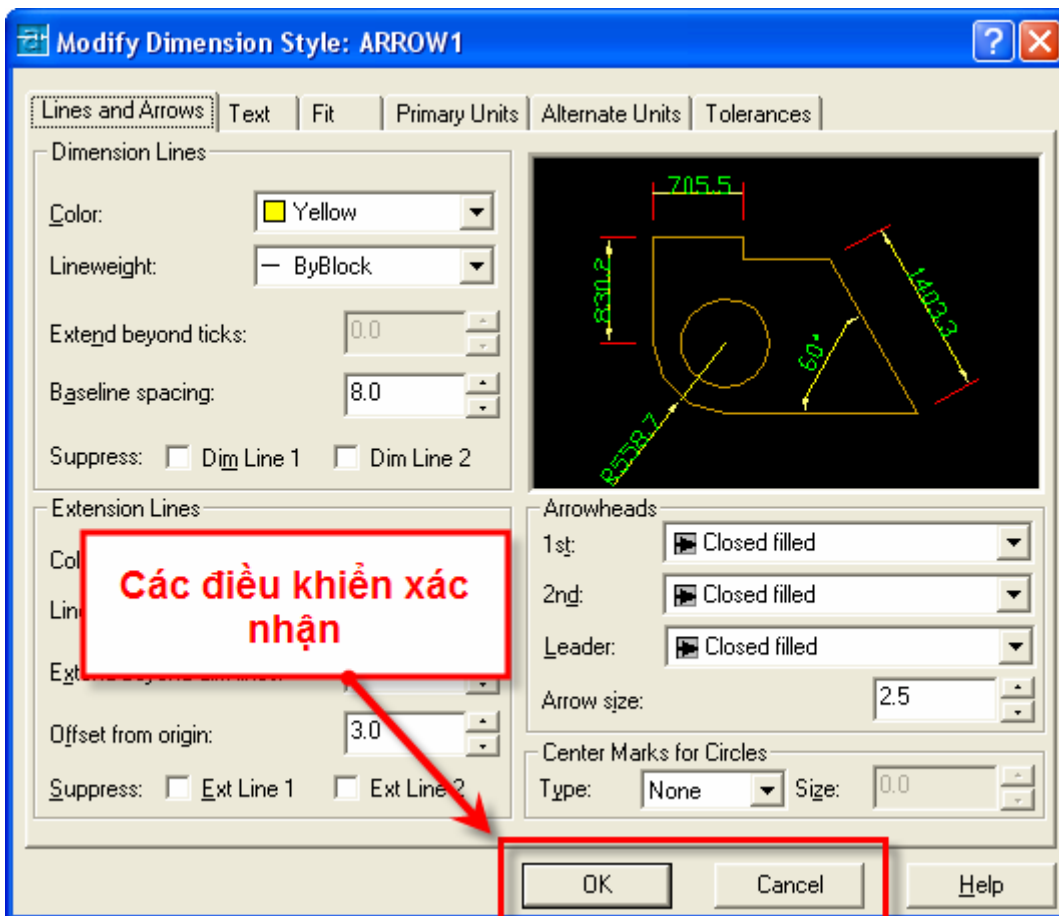
#### Để tạo ra UserForm, làm theo trình tự sau:

1. Xác định sự cần thiết phải tạo giao diện nhập-xuất dữ liệu riêng: Giao diện do ứng dụng nền cung cấp không đủ hoặc không thích hợp cho việc nhập dữ liệu hoặc xuất kết quả của chương trình.
2. Xác định cách thức và trình tự tương tác của người sử dụng trên giao diện: để có thể bố trí các điều khiển sao cho thuận tiện đối với người dùng, ví dụ như theo thói quen điều khiển của đa số người sử dụng là *từ trái sang phải, từ trên xuống dưới*.
3. Xác định số lượng UserForm cần phải tạo cho quá trình nhập dữ liệu cũng như việc hiển thị kết quả: chỉ nên sử dụng vừa đủ và phân theo chủ đề của công việc, ví dụ nên phân tách giao diện nhập dữ liệu với giao diện trình bày kết quả và các điều khiển (nút bấm) khác.
4. Xác định các loại dữ liệu cần nhập vào, các dữ liệu theo tình huống và các minh họa bằng hình ảnh kèm theo để giải thích rõ cho người sử dụng ý nghĩa của các thông số cần được nhập vào. Căn cứ vào các loại dữ liệu cần nhập trên để xác định các thành phần điều khiển phù hợp và đưa vào UserForm tương ứng. Cần chú ý rằng, các điều khiển, ngoài việc đáp ứng yêu cầu về mặt chức năng, chúng cũng cần được trình bày và giải thích một cách dễ hiểu và có tính thẩm mỹ.



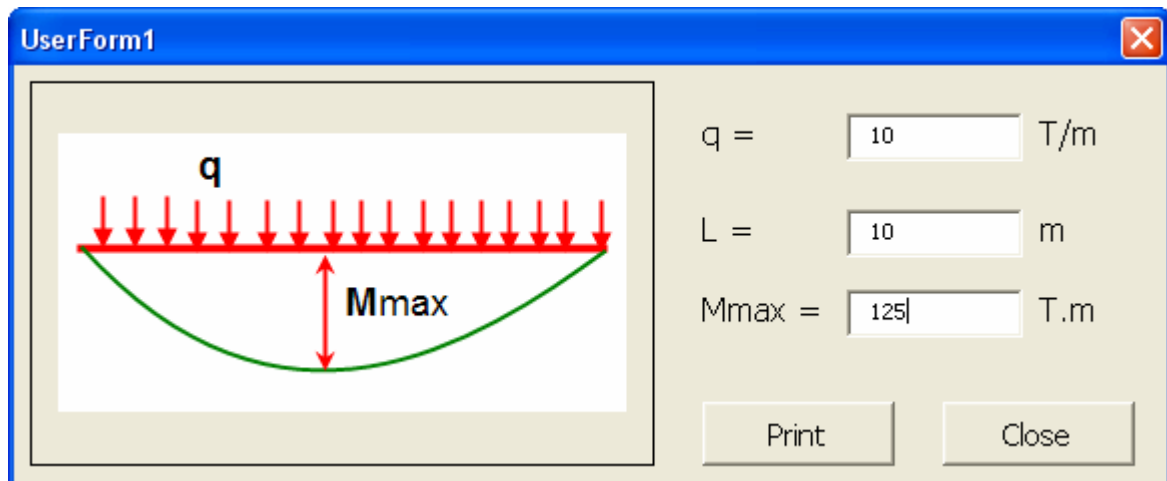
Hình III-17: Ý nghĩa các loại dữ liệu cần nhập vào được minh họa bằng hình ảnh.

5. Lựa chọn các điều khiển phục vụ cho việc xác nhận dữ liệu sau khi nhập xong hoặc ra lệnh cho quá trình xử lý các dữ liệu này bắt đầu thực hiện. Thông thường các điều khiển này là hệ thống các nút bấm (Button) để xác nhận các dữ liệu đã nhập xong, yêu cầu bắt đầu xử lý hoặc hủy bỏ các dữ liệu đã nhập.



Hình III-18: Bố trí các điều khiển trên UserForm.

6. Lựa chọn hình thức hiển thị kết quả từ đó lựa chọn các thành phần điều khiển phù hợp, ví dụ như kết quả tính toán là số hoặc hình vẽ thì cần chọn điều khiển thích hợp để trình bày.



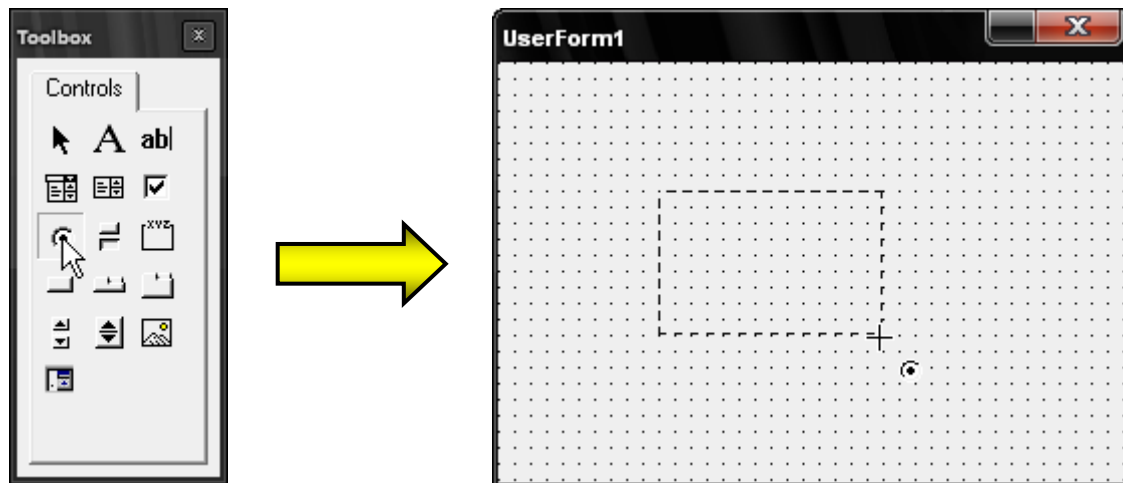
Hình III-19: Trình bày kết quả bằng điều khiển hỗ trợ văn bản và hình ảnh.

7. Viết mã lệnh cho các thành phần điều khiển. Mã lệnh này sẽ được lưu trữ trong phần code của UserForm.

#### 11.1.1. Tạo UserForm và các thành phần điều khiển trong VBA IDE

Trong VBA IDE, UserForm được tạo ra bằng cách chọn trình đơn **Insert** ⇒ **UserForm**

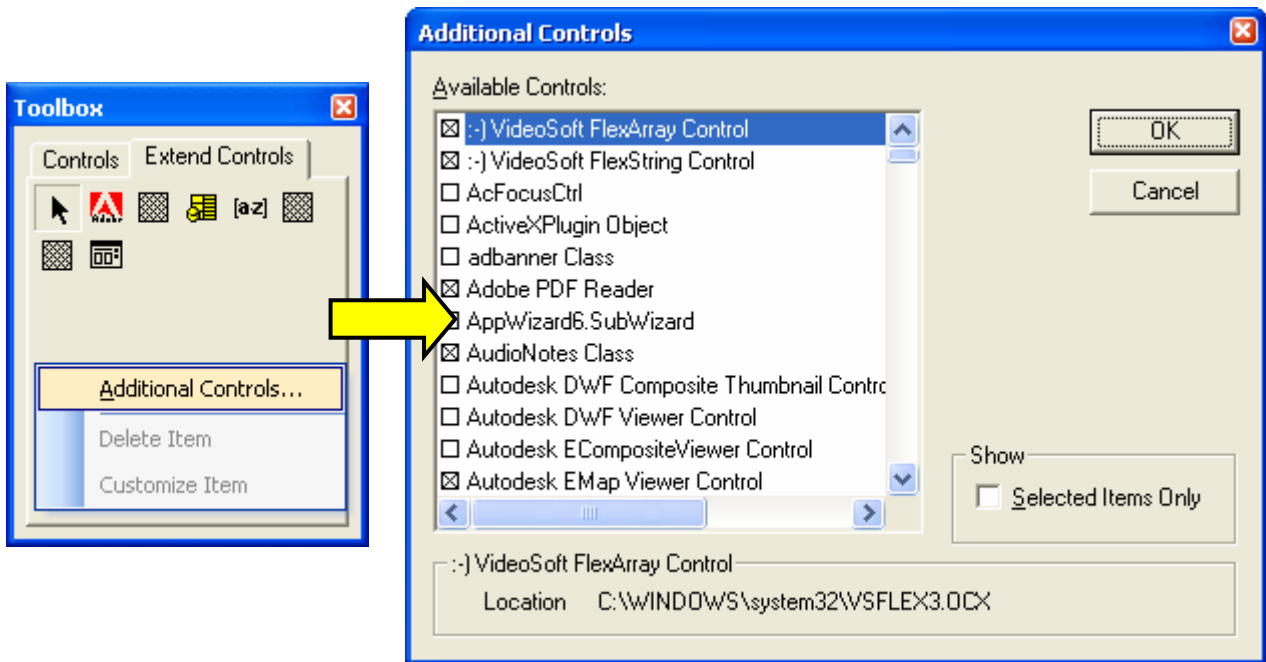
Sau khi tạo UserForm, ta có thể thêm các thành phần điều khiển vào UserForm bằng cách lựa chọn điều khiển cần dùng từ hộp công cụ điều khiển (Control Toolbox) và thực hiện thao tác *kéo/thả* vào vị trí thích hợp UserForm. Kích thước của điều khiển có thể thay đổi một cách dễ dàng nhờ thao tác kéo chuột ở vùng biên của chúng.



Điều khiển được lựa chọn trên Control Toolbox

Điều khiển được tạo bằng cách kéo/thả vào UserForm

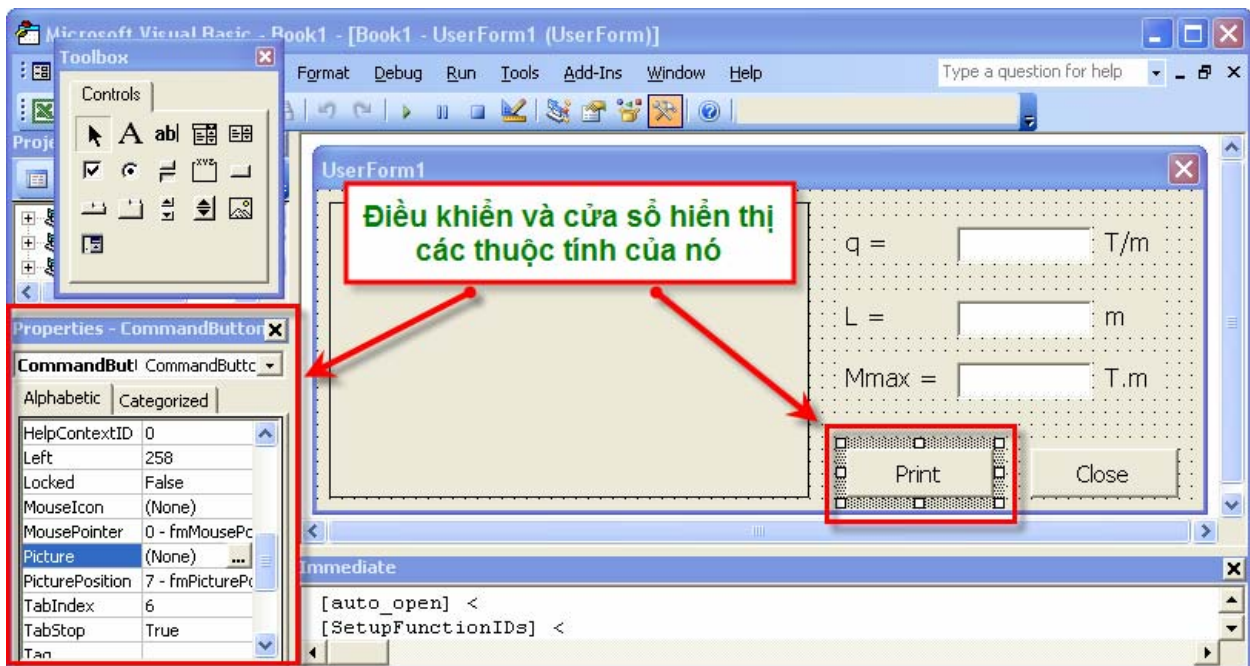
Thông thường trong hộp công cụ mặc định của VBA IDE chỉ có các thành phần điều khiển chuẩn của VB, các điều khiển này đáp ứng được hầu hết các nhu cầu cơ bản về thiết kế giao diện. Tuy nhiên người dùng có thể bổ sung những thành phần điều khiển khác vào hộp công cụ trên bằng cách sử dụng **Additional Controls** có sẵn trên hộp công cụ (hiển thị bằng cách nhấn chuột phải vào hộp công cụ). Với mỗi máy tính khác nhau thì nội dung các điều khiển có thể bổ sung là khác nhau bởi chúng phục thuộc vào các thư viện lập trình được cài đặt trên máy tính đó.



**Hình III-20: Bổ sung thêm điều khiển cho hộp công cụ (Toolbox) của VBA IDE.**

**11.1.2. Các thuộc tính của UserForm và các thành phần điều khiển.**

Các thuộc tính (Properties) là các thông số quy định đặc điểm, tính chất cũng như trạng thái của UserForm hay các điều khiển, ví dụ màu nền của một điều khiển được quy định bởi thuộc tính BackColor. Những thuộc tính này có thể được thay đổi trong lúc thiết kế UserForm hoặc lúc chương trình đang chạy. Tuy nhiên một số thuộc tính không cho phép thay đổi mà chỉ cho phép người dùng biết được giá trị của nó (thuộc tính chỉ đọc – Read Only). Trong quá trình thiết kế UserForm, khi ta dùng chuột chọn bất cứ thành phần nào trên UserForm (kể cả chính UserForm) thì các thuộc tính của nó sẽ được hiển thị tương ứng trong cửa sổ **Properties** của VBA IDE.



**Hình III-21: Thành phần điều khiển và vị trí hiển thị các thuộc tính của nó.**

Một số thuộc tính cơ bản của UserForm và các điều khiển:

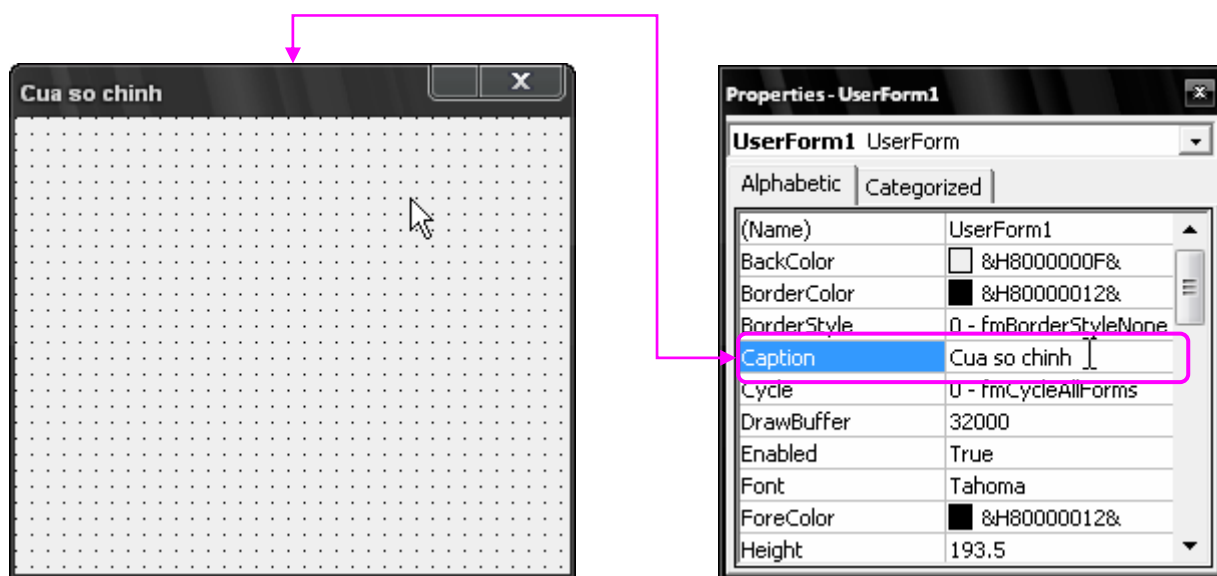


Thuộc tính	Giải thích
Name	Thẻ hiện tên của UserForm hay điều khiển. Đây là thuộc tính rất quan trọng, là yếu tố xác định điều khiển khi lập trình. Thuộc tính này chỉ được thay đổi lúc thiết kế giao diện (trong cửa sổ Properties của VBA IDE).
BackColor	Giá trị kiểu Long thể hiện màu nền của UserForm hay điều khiển.
Caption	Giá trị kiểu String thể hiện tiêu đề của UserForm hay điều khiển.
Enable	Giá trị kiểu logic (Boolean) xác định trạng thái làm việc của điều khiển, giá trị bằng True ứng với trạng thái hoạt động, giá trị bằng False ứng với trạng thái không hoạt động (điều khiển coi như bị vô hiệu hoá và thường được hiển thị mờ đi trên UserForm).
Visible	Giá trị kiểu logic (Boolean) xác định trạng thái hiển thị của điều khiển, giá trị bằng True ứng với sự hiển thị điều khiển, giá trị bằng False ứng với sự ẩn điều khiển.
Font	Thẻ hiện kiểu và cỡ chữ hiển thị trên UserForm hoặc điều khiển.
Picture	Thẻ hiện hình ảnh trên nền UserForm hoặc điều khiển.
ControlTipText	Giá trị kiểu String thể hiện chú thích về điều khiển khi chuột di chuyển qua (Tooltip) trong lúc chương trình hoạt động.
Mouselcon	Thẻ hiện biểu tượng con trỏ chuột hiển thị trên điều khiển.
MousePointer	Thẻ hiện loại con trỏ chuột hiển thị trên nút lệnh.

**CỢI Ý** Ngoài ra, ứng với mỗi loại điều khiển có thể còn có thêm nhiều thuộc tính khác hoặc không có một số các thuộc tính được liệt kê ở trên. Người dùng có thể tìm hiểu các thuộc tính này trong Object Browser hoặc trong Help (chọn điều khiển và bấm F1) của VBA IDE.

Việc thay đổi thuộc tính của các điều khiển có thể được thực hiện bằng hai cách:

- Cách 1:** Thay đổi trực tiếp trong quá trình thiết kế: chọn điều khiển và thay đổi giá trị của các thuộc tính trong cửa sổ **Properties** của VBA IDE. Cách này trực quan và dễ thực hiện đối với đa số các thuộc tính của hầu hết các điều khiển. Ví dụ: để thay đổi tiêu đề cho một UserForm dưới đây, kích chuột chọn UserForm sau đó nhập tên của tiêu đề vào phần Caption của cửa sổ Properties.



Hình III-22: Thay đổi giá trị thuộc tính trong khi thiết kế UserForm.

2. **Cách 2:** Thay đổi trong lúc chương trình đang chạy: về thực chất, các thuộc tính chính là dữ liệu của các thành phần điều khiển (thường gọi chung các điều khiển này là đối tượng) hay chính là các biến được định nghĩa riêng cho điều khiển đó cho nên ta có thể sử dụng phép gán thông thường để thay đổi giá trị cho một số thuộc tính. Cú pháp thực hiện như sau:

```
<Tên_điều_khiển>.<Tên_thuộc_tính> = giá trị thuộc tính  
<Tên_UserForm>.<Tên_thuộc_tính> = giá trị thuộc tính
```

**GỢI Ý** Tên\_điều\_khiển hay Tên\_UserForm ở đây chính là giá trị thuộc tính Name của điều khiển đã được đặt khi thiết kế. Khi viết mã lệnh trong một UserForm thì có thể thay việc dùng tên của UserForm đó bằng từ khoá **Me**.

Ví dụ, ứng với UserForm có tên là UserForm1 như ở trên, có thể thay đổi tiêu đề của nó bằng mã lệnh như sau:

```
UserForm1.Caption = "Cua so chinh"
```

### 11.1.3. Các phương thức của UserForm và các thành phần điều khiển.

Các phương thức có thể xem chúng là những chương trình con đặc biệt, chúng chỉ làm việc với các dữ liệu của điều khiển và tương tác lên chính điều khiển đó. Để phương thức hoạt động, cần phải gọi nó (tương tự như gọi chương trình con) bằng mã lệnh khi lập trình. Cú pháp gọi phương thức của một điều khiển hay UserForm cũng tương tự như với biến đối tượng:

```
<Tên_điều_khiển>.<Tên_phương_thức> <(tham_số_của_phương_thức)>  
<Tên_UserForm>.<Tên_phương_thức> <(tham_số_của_phương_thức)>
```

Ví dụ: muốn hiển thị UserForm1 như ở trên, gọi phương thức Show của nó với mã lệnh như sau:

```
UserForm1.Show
```

Trong phạm vi của giáo trình, không thể liệt kê tất cả các phương thức của các điều khiển. Trong phần sau sẽ trình bày một số các phương thức cơ bản của một số loại điều khiển thông dụng. Để biết chi tiết về những phương thức khác, có thể tra cứu trong Object Browser hoặc trong Help của VBA IDE

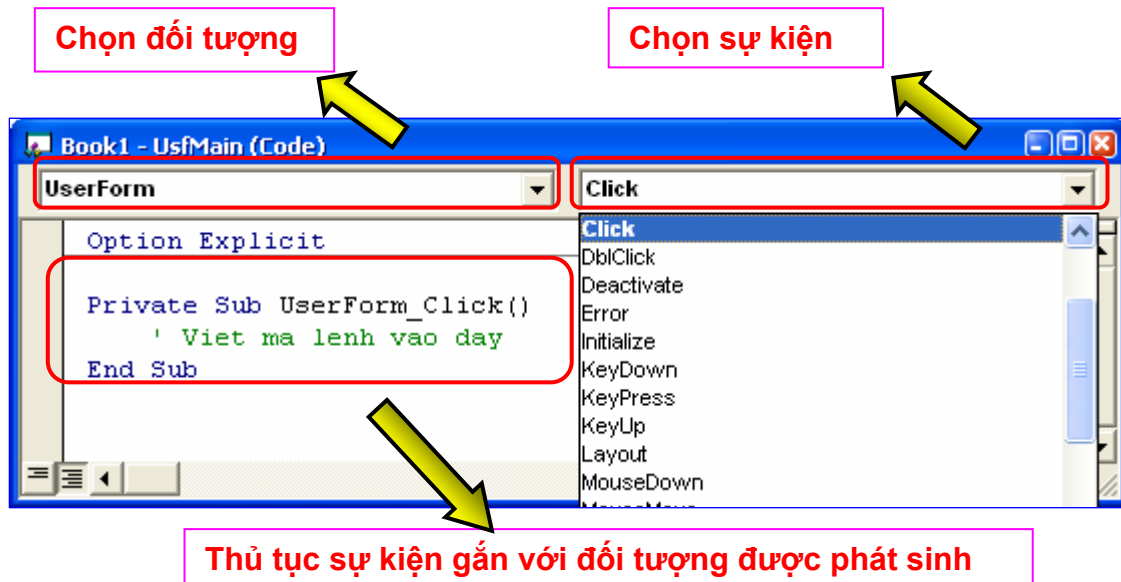
### 11.1.4. Các sự kiện trên giao diện.

Các sự kiện trên UserForm hoặc các điều khiển được phát sinh khi có một hoạt động nào đó xảy ra – thường được phát sinh từ phía người dùng (sự kiện cũng có thể được phát sinh một cách gián tiếp từ quá trình thực hiện một phương thức nào đó). Ví dụ, khi người dùng rê chuột trên bề mặt UserForm sẽ phát sinh sự kiện MouseMove, khi người dùng kích chuột trên UserForm sẽ phát sinh sự kiện Click.

Đi cùng với sự kiện còn có **thủ tục sự kiện**: là chương trình được thi hành khi sự kiện xảy ra.

Thủ tục sự kiện cho phép người lập trình xử lý các tương tác của người dùng trên giao diện bằng cách viết các mã lệnh trong thủ tục sự kiện.

Để viết mã lệnh cho một thủ tục sự kiện trên một UserForm, vào cửa sổ mã lệnh của UserForm đó (nháy đúp chuột vào UserForm), chọn điều khiển và loại sự kiện tương ứng. Sau đó viết mã lệnh vào trong thủ tục sự kiện đã được tạo ra.



Một số sự kiện cơ bản của UserForm và các điều khiển:

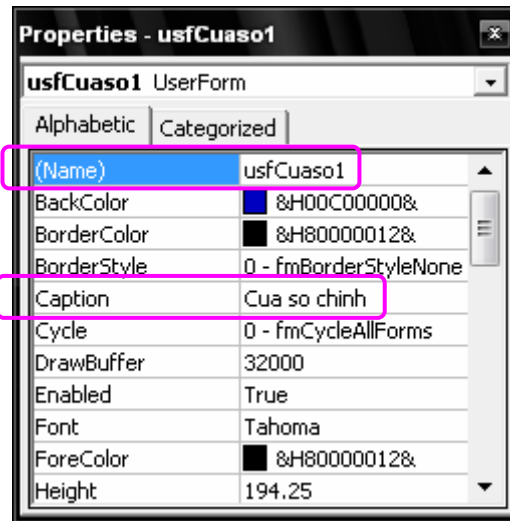
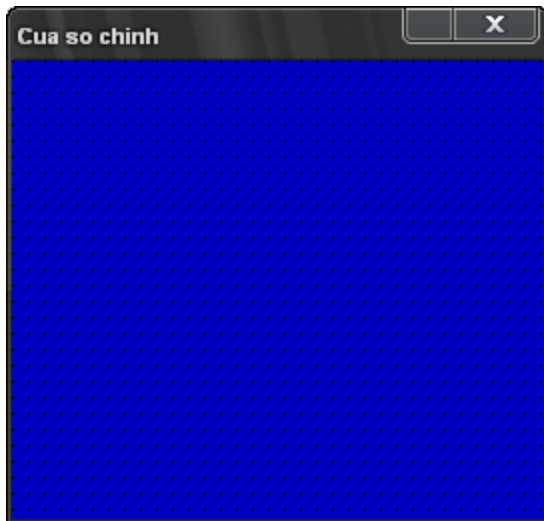
Sự kiện	Giải thích
Click	xảy ra khi người dùng kích chuột trên UserForm hoặc trên điều khiển
DbClick	xảy ra khi người dùng kích đúp chuột trên UserForm hoặc trên điều khiển
KeyPress	xảy ra khi người dùng nhấn một phím
KeyUp	xảy ra khi người dùng nhả một phím (sau khi đã nhấn xuống)
KeyDown	xảy ra khi người dùng nhấn một phím (nhưng chưa nhả ra)
MouseMove	xảy ra khi người dùng rê chuột ngang qua một điều khiển hoặc trên UserForm
MouseUp	xảy ra khi người dùng nhả phím chuột (sau khi đã nhấn chuột)
MouseDown	xảy ra khi người dùng nhấn phím chuột (nhưng chưa nhả ra)

### 11.1.5. Ví dụ

Tạo một UserForm và viết mã lệnh để khi kích chuột vào UserForm sẽ hiển thị số lần kích chuột trên tiêu đề của nó đồng thời đổi màu nền của UserForm theo tình huống: nếu số lần kích chuột là chẵn thì màu đen, là lẻ thì màu trắng.

Các thao tác như sau:

1. Thêm UserForm vào trong dự án bằng cách chọn Insert ⇒ UserForm.
2. Đặt tên UserForm là "usfCuaso1" trong thuộc tính Name của cửa sổ Properties; đặt tiêu đề xuất phát của UserForm là "Cua so chinh" trong thuộc tính Caption của cửa sổ Properties.



3. Viết mã lệnh cho sự kiện Click của UserForm (hiển thị cửa sổ lệnh của UserForm bằng cách nhấp đúp chuột vào UserForm, chọn UserForm và sự kiện Click).

Mã lệnh cho thủ tục sự kiện Click như sau:

```
Private Sub UserForm_Click()
    Static numClick As Long
    numClick = numClick + 1
    If numClick Mod 2 = 0 Then
        Me.BackColor = vbBlack
    Else
        Me.BackColor = vbWhite
    End If
    usfCuaso1.Caption = "Number of Click: " & Str(numClick)
End Sub
```

**GỢI Ý** Trong đoạn mã trên, vbBlack là hằng số tương ứng với màu đen, vbWhite là hằng số tương ứng với màu trắng. Hai hằng số này được định nghĩa sẵn trong VB.

### 11.2. Làm việc với UserForm

Các nguyên tắc làm việc với UserForm như thiết lập và thay đổi thuộc tính, gọi các phương thức hay xử lý các sự kiện đã được trình bày ở phần trước. Dưới đây chỉ giới thiệu một số phương thức khác của UserForm.

- ◆ **Hiển thị UserForm:** thực hiện phương thức Show

```
Tên_UserForm.Show [vbModal/ vbModeless]
```

Nếu dùng vbModal (hoặc 1): hộp thoại (UserForm) sẽ hiển thị ở dạng Modal – tức là luôn tiếp nhận tương tác người dùng với hộp thoại, người dùng chỉ có thể chuyển hướng tương tác sang nơi khác khi đóng hộp thoại. Đây là kiểu hiển thị mặc định của hộp thoại.

Nếu dùng vbModeless (hoặc 0): hộp thoại vẫn được hiển thị nhưng người dùng có thể chuyển hướng tương tác sang nơi khác mà không cần đóng hộp thoại.

- ◆ **Ẩn UserForm:** gọi phương thức Hide

```
Tên_UserForm.Hide
```

- ◆ Quay lại trạng thái trước lệnh cuối cùng được thực hiện trên UserForm: thực hiện phương thức UndoAction

Tên\_UserForm.UndoAction

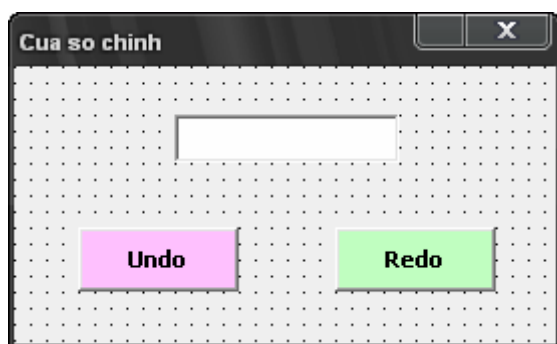
- ◆ Trả lại trạng thái trước khi thực hiện Undo: thực hiện phương thức RedoAction

Tên\_UserForm.RedoAction

**CHÚ Ý** Với chương trình sử dụng nhiều UserForm, để tránh nhầm lẫn trong khi sử dụng chương trình, chỉ nên hiển thị UserForm cần dùng còn những UserForm khác thì ẩn đi. Trước khi gọi phương thức Show của UserForm cần hiển thị, phải ẩn UserForm không dùng đến bằng phương thức Hide của nó.

### Ví dụ

Tạo một UserForm với các điều khiển như hình dưới đây:



Trình tự thực hiện như sau:

1. Thêm một UserForm vào dự án.
2. Chọn vào UserForm vừa tạo, chọn biểu tượng **abl** trong hộp công cụ điều khiển (Control Toolbox). Sau đó, rê thả chuột trên UserForm để tạo một hộp văn bản (TextBox).
3. Tiếp tục chọn UserForm trên, chọn biểu tượng **▭** trong hộp công cụ điều khiển, rê thả chuột để tạo một nút lệnh (Command Button), đặt tên (thuộc tính Name) của nút lệnh là cmdUndo, đặt tiêu đề (thuộc tính Caption) của nút lệnh là Undo.
4. Tương tự như trên tạo nút lệnh cmdRedo với tiêu đề Redo.
5. Viết các thủ tục sự kiện Click cho các nút lệnh trên như sau:

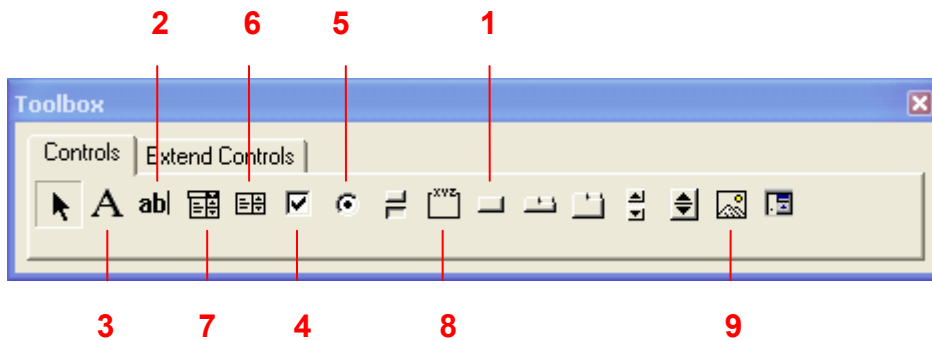
```
Private Sub cmdRedo_Click()
    Me.RedoAction
End Sub
Private Sub cmdUndo_Click()
    Me.UndoAction
End Sub
```

Sau đó, chọn UserForm và nhấn phím F5 để chạy chương trình. Nhập một dòng văn bản vào trong hộp văn bản. Kích chuột vào nút Undo, sau đó là nút Redo và theo dõi kết quả.

### 11.3. Các điều khiển thông dụng

Theo mặc định, trên Toolbox có sẵn một số điều khiển thông dụng trong thẻ Control, những điều khiển này đáp ứng được hầu hết nhu cầu thiết kế giao diện thông thường trên UserForm.

Trong những phần trước đã nhắc nhiều đến việc sử dụng các điều khiển trên Toolbox nhưng chưa có tính hệ thống vì vậy phần này sẽ trình bày những nội dung cơ bản để có thể sử dụng một cách hiệu quả các điều khiển này.



Hình III-23: Các điều khiển cơ bản theo mặc định trong VBA IDE

## 1 Nút lệnh (Command Button)

- Command Button thường được dùng để thực hiện một quyết định nào đó từ phía người dùng (thông qua việc kích chuột vào Command Button hoặc nhấn Enter).
- Command Button nên có thuộc tính `Caption` (tiêu đề) và `Picture` (hình ảnh) phản ánh đúng tính năng mà nó đảm nhận. Sự kiện hay được gọi khi sử dụng Command Button là sự kiện `Click` hoặc `DblClick` (kích đúp chuột).
- Để thay đổi vị trí của Command Button trong khi chạy chương trình, sử dụng phương thức `Move`

```
Tên_Command Button.Move [Left ], [Top], [Width], [Height]
```

Trong đó các tham số thể hiện vị trí góc trái trên (`left`, `top`) và kích thước (`Width`, `Height`) mới của Command Button sau khi di chuyển.

- Để thiết lập trạng thái nhận lệnh (nhận tiêu điểm – *focus*), sử dụng phương thức `SetFocus`

```
Tên_Command Button.SetFocus
```

Các phương thức `Move` và `SetFocus` như trên không chỉ áp dụng đối với Command Button, mà còn được áp dụng với đa số các điều khiển khác.

## 2 Hộp văn bản (TextBox)

TextBox được dùng để nhập dữ liệu dạng văn bản (`text`) và nó được điều khiển bằng cách thiết lập những thuộc tính hay sự kiện hoặc sử dụng các phương thức phù hợp. Dưới đây là một số thành phần chính dùng để điều khiển TextBox:

- Các thuộc tính dùng để thiết lập cách thức hiển thị cho TextBox

Thuộc tính	Mô tả	Ghi chú
<code>Text</code>	Nội dung văn bản chứa trong điều khiển.	Kiểu String
<code>TextAlign</code>	Cách thức hiển thị văn bản trong điều khiển	Tham khảo Object Browser
<code>MaxLength</code>	Qui định độ dài tối đa của văn bản trong điều khiển (Nếu đặt <code>MaxLength=0</code> , độ dài của văn bản là tùy ý)	Kiểu Long

MultiLine	Hiển thị nhiều dòng hay một dòng	Kiểu Boolean
ScrollBars	Hiển thị thanh cuộn ngang hay dọc nếu nội dung văn bản lớn hơn kích thước của điều khiển	Tham khảo Object Browser

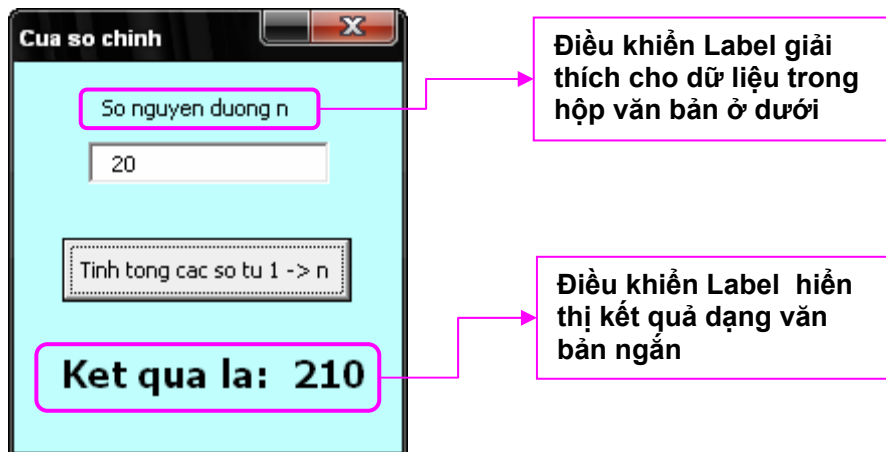
✦ Các phương thức hỗ trợ việc nhập văn bản vào TextBox

Phương thức	Mô tả	Ghi chú
Copy	Chép nội dung được đánh dấu trong điều khiển vào bộ nhớ đệm	Tham khảo trong Object Browser hoặc Help
Cut	Di chuyển nội dung được đánh dấu trong điều khiển vào bộ nhớ đệm	
Paste	Chép nội dung từ bộ nhớ đệm vào điều khiển	

✦ Các sự kiện: thường dùng để xử lý khi có tác động lên TextBox, thường sử dụng hai sự kiện là: `KeyPress` và `Change`. Sự kiện, về bản chất là một chương trình con dạng Sub và được tự động gọi ra tương ứng với tác động nào đó lên TextBox, ví dụ như bấm phím hay thay đổi nội dung. Sự kiện `Change` được gọi khi nội dung văn bản trong TextBox bị thay đổi. Còn sự kiện `KeyPress` được gọi khi có một phím được nhấn khi con trỏ đang nằm trong điều khiển. Sự kiện `KeyPress` có một tham số là `KeyAscii`. Tham số này có kiểu `Integer` và chứa mã ASCII của phím được nhấn (để biết mã ASCII của các phím, tham khảo `KeyCodeConstants` trong Object Browser).

### 3 Nhãn (Label)

Label thường được sử dụng để hiển thị một văn bản ngắn gọn trên UserForm hoặc dùng kèm với một điều khiển nào đó trên UserForm với mục đích là giải thích ý định sử dụng cho điều khiển đó. Nội dung văn bản trong Label được thiết lập hoặc thay đổi thông qua thuộc tính `Caption` của nó. Tương tác với thuộc tính `Caption` của Label cũng tương tự như đối với thuộc tính `Caption` của tất cả các điều khiển khác và đã được trình bày ở các phần trước.



### 4 Hộp đánh dấu (CheckBox)

CheckBox thường được sử dụng để lựa chọn thông tin phù hợp trong một danh sách các thông tin liên quan được liệt kê hoặc dùng để bổ sung nội dung cho một dữ liệu nào đó.

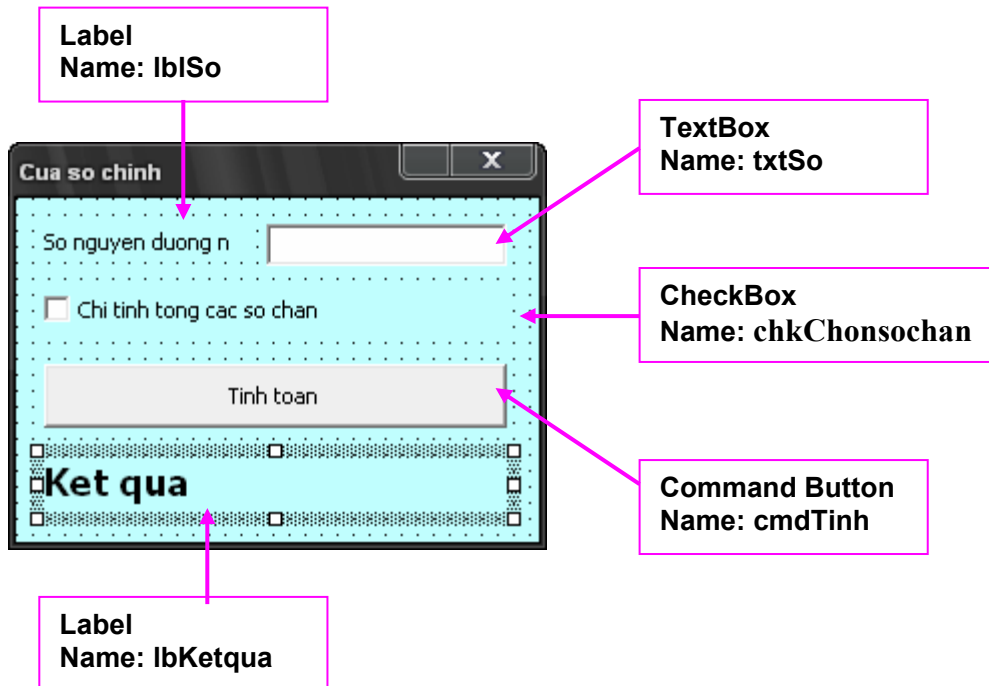
Để thiết lập hay đọc trạng thái của CheckBox (được chọn hay không được chọn), sử dụng thuộc tính `Value`. Thuộc tính này có kiểu `Boolean`, nếu giá trị của nó là `True` thì có nghĩa là CheckBox được chọn, giá trị là `False` nghĩa là CheckBox không được chọn.

Ví dụ: tạo giao diện nhập dữ liệu và tính tổng các số từ 1 đến n với tùy chọn bằng CheckBox như sau:

## GIÁO TRÌNH TỰ ĐỘNG HOÁ THIẾT KẾ CẦU ĐƯỜNG

- ◆ Nếu không chọn CheckBox (không đánh dấu) thì tính tổng của các số từ 1 đến n.
- ◆ Nếu chọn CheckBox (đánh dấu) thì chỉ tính tổng các số chẵn trong khoảng từ 1 đến n.

Thiết kế UserForm với các thành phần như hình dưới đây:



Mã lệnh cho thủ tục sự kiện Click cho nút lệnh cmdTinh như sau:

```
Private Sub cmdTinh_Click()  
    Dim i As Long  
    Dim so As Long  
    so = CLng(txtSo.Text) ' chuyen doi du lieu tu txtSo vao bien so  
    Dim tong As Double  
    tong = 0  
    Dim buocnhay As Long ' buoc nhay cua bien chay  
    If chkChonsochan.Value Then ' xet lua chon chi tinh tong so chan  
        buocnhay = 2  
    Else  
        buocnhay = 1  
    End If  
    For i = 0 To so Step buocnhay  
        tong = tong + i  
    Next  
    lblKetqua.Caption = "Ket qua: " & Str(tong) ' hien thi ket qua  
End Sub
```

### 5 Nút tùy chọn (OptionButton)

OptionButton thường được dùng để yêu cầu người dùng chọn một trong các thông tin được liệt kê sẵn. Để tạo nhóm các OptionButton, ta có thể đặt chúng trong một điều khiển khung (Frame) hoặc đặt chúng trên UserForm.





Hai OptionButton phục vụ cho việc lựa chọn loại hình dự án. Người dùng chỉ có thể chọn một trong hai điều khiển này.

Để thiết lập trạng thái chọn hay không chọn cho OptionButton, sử dụng thuộc tính Value, thuộc tính này có kiểu Boolean. Nếu giá trị của nó là True thì có nghĩa là OptionButton đó được chọn, còn nếu giá trị là False thì OptionButton đó không được chọn. Ví dụ mã lệnh sau tương đương với việc người dùng chọn OptionButton tên là optDAmoi

```
optDAmoi.Value=True
```

## 6 Hộp danh sách (ListBox)

ListBox cho phép liệt kê một danh sách các giá trị để người dùng có thể quan sát và lựa chọn một hoặc một vài giá trị trong danh sách này.

Mỗi giá trị trong ListBox luôn có chỉ số (Index) và nội dung (Text).

◆ Thuộc tính:

Thuộc tính	Mô tả	Ghi chú
List	Trả về danh sách các giá trị trong điều khiển	Tham khảo Object Browser
ListCount	Trả về số lượng các giá trị trong danh sách	Kiểu Long
ListIndex	Trả về chỉ số của giá trị được chọn trong danh sách	Kiểu Variant
Text	Trả về nội dung của giá trị được chọn	Kiểu String
Selected(i)	Kiểm tra xem phần giá trị i có được chọn hay không.	Kiểu Boolean

◆ Phương thức:

Phương thức	Mô tả	Ghi chú
AddItem	Thêm một giá trị vào trong danh sách	Tham khảo trong Object Browser hoặc Help
RemoveItem	Xoá một giá trị khỏi danh sách	
Clear	Xoá toàn bộ danh sách	

Ví dụ: tạo UserForm với ListBox như hình dưới:



UserForm: usfMc

ListBox: lstDsMc

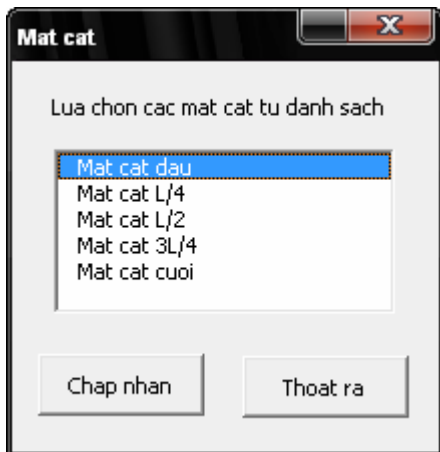
**Yêu cầu:** khi chương trình hoạt động, trong ListBox sẽ có một danh sách các mặt cắt được thống kê ra.

Mã lệnh cho sự kiện Initialize (sự kiện này được tự động gọi khi chương trình nạp UserForm vào bộ nhớ của máy tính) của UserForm như sau:

```
Private Sub UserForm_Initialize()
    lstDsMc.AddItem "Mat cat dau", 0
    lstDsMc.AddItem "Mat cat L/4", 1
    lstDsMc.AddItem "Mat cat L/2", 2
    lstDsMc.AddItem "Mat cat 3L/4", 3
    lstDsMc.AddItem "Mat cat cuoi", 4
End Sub
```

**GỢI Ý** Phương thức AddItem có hai tham số đều là tham số tùy chọn: tham số thứ nhất là nội dung của phần tử, tham số thứ hai là vị trí chèn phần tử trong danh sách.

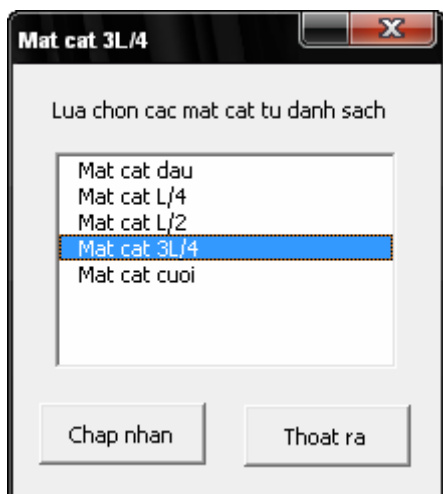
Kết quả chạy chương trình như sau:



Để biết người dùng đã lựa chọn phần tử nào trong ListBox, viết mã lệnh cho sự kiện Click của ListBox như sau:

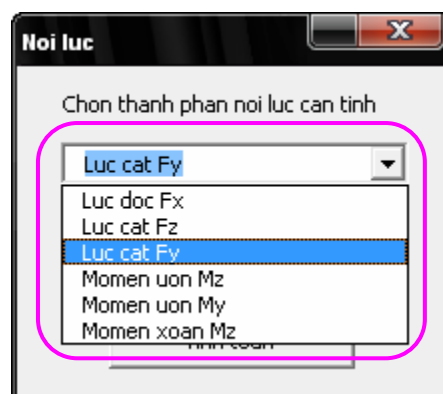
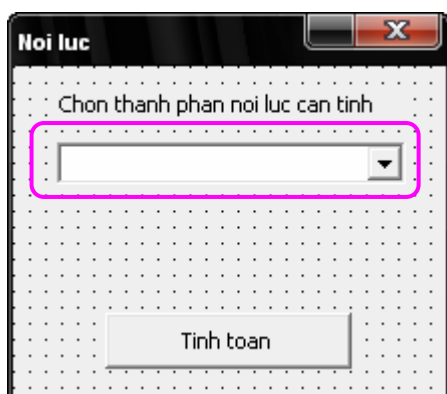
```
Private Sub lstDsMc_Click()
    `Hiện thị giá trị được chọn lên tiêu đề của UserForm
    Me.Caption = lstDsMc.Text
End Sub
```

Kết quả khi người dùng chọn một giá trị trong ListBox:



### 7 Hộp danh sách tổ hợp (ComboBox)

Tương tự như ListBox, nhưng danh sách các giá trị được thể hiện theo kiểu hiện ra đầy đủ khi người dùng kích chuột vào. Ngoài ra điều khiển này còn cho phép người dùng nhập giá trị cần tìm vào, điều này giúp cho việc lựa chọn được nhanh hơn khi người dùng biết tên giá trị cần chọn trong danh sách và chiều dài của danh sách lại quá lớn.

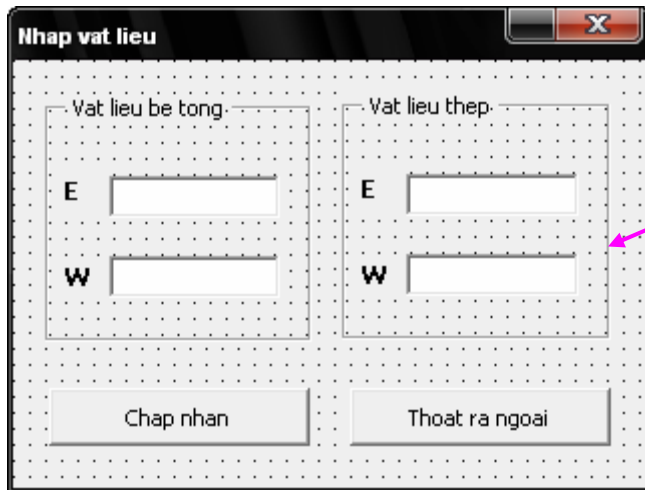


Các thuộc tính và phương thức của ComboBox tương tự như ListBox.

### 8 Khung (Frame)

Frame cho phép nhóm các điều khiển trên UserForm lại theo chủ đề, giúp cho việc trình bày trên UserForm được rõ ràng và giúp cho người dùng dễ sử dụng chương trình. Frame còn được dùng để tập hợp các OptionButton thành một nhóm.

Khi tạo nhóm điều khiển trong Frame, nên tạo Frame trước rồi mới tạo các điều khiển thành phần trong nó (khi đó các điều khiển tạo sau sẽ được gắn và trong Frame).

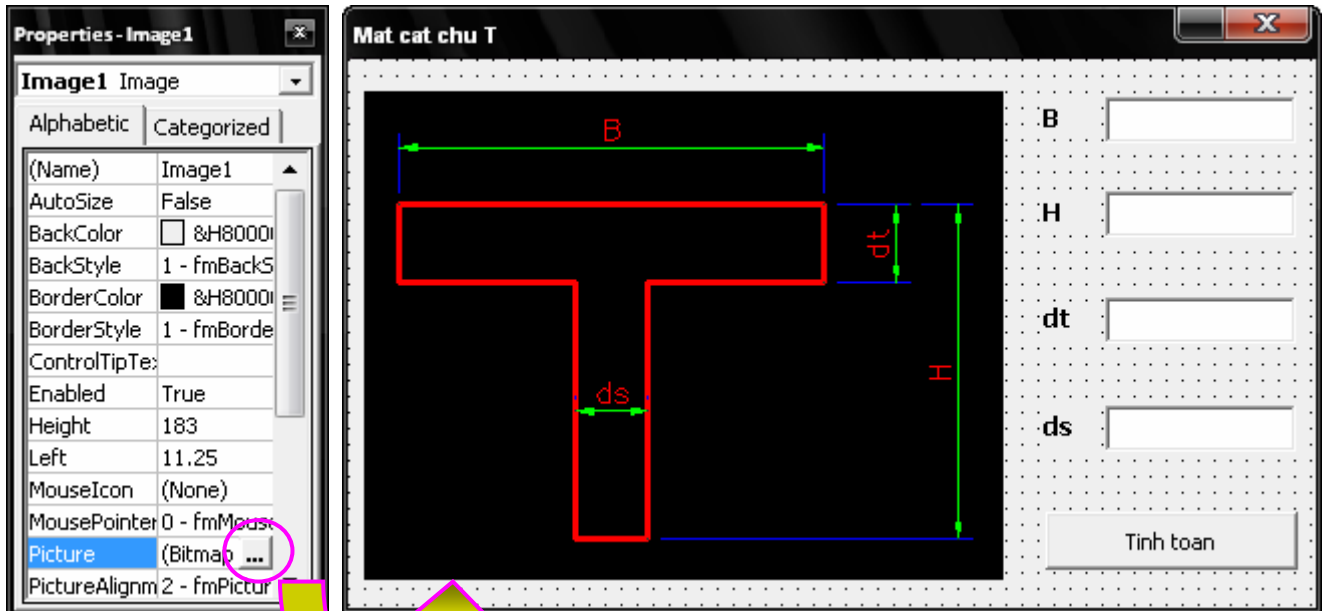


Frame nhóm các điều khiển phục vụ việc nhập dữ liệu cho các nhóm vật liệu khác nhau.

**9** Hình ảnh (Image)

Image cho phép hiển thị hình ảnh trên một vùng của UserForm. Sử dụng điều khiển này giúp cho việc minh họa dữ liệu cần nhập trở nên rõ ràng và dễ hình dung.

Để chèn hình ảnh vào trong điều khiển, sử dụng thuộc tính Picture của nó.



**12. Các hộp thoại thông dụng**

**12.1. Hộp thông điệp (Message Box – MsgBox)**

MsgBox được sử dụng để nhắc nhở, thông báo hoặc cảnh báo người dùng. MsgBox có thể được gọi theo kiểu thủ tục (không có giá trị trả về), hoặc theo kiểu hàm (giá trị trả về là nút lệnh được người dùng chọn).

Cú pháp gọi MsgBox như sau:

- ◆ Dạng thủ tục:

```
MsgBox Prompt, [Buttons], [Title]
```

◆ Dạng hàm

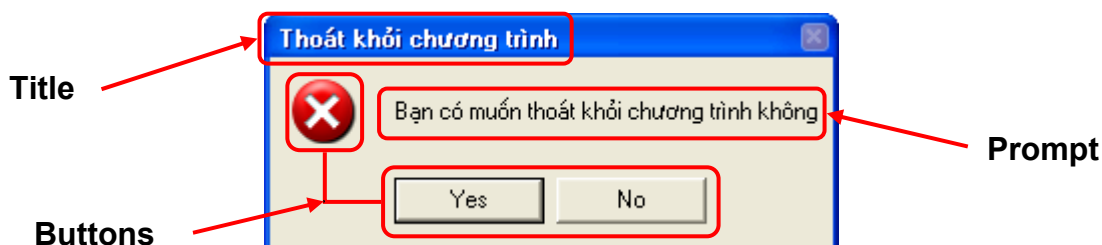
**MsgBox(Prompt, [Buttons], [Title])**

(Giá trị trả về của hàm có kiểu `VbMsgBoxResult` – tham khảo trong Object Browser)

Tham số	Mô tả
Prompt	Kiểu String. Nội dung dòng nhắc trong MsgBox.
Buttons	Kiểu <code>VbMsgBoxStyle</code> (tham khảo trong Object Browser). Kiểu hiển thị biểu tượng và nút lệnh trong MsgBox.
Title	Kiểu String. Nội dung dòng tiêu đề của MsgBox.

Ví dụ: MsgBox được gọi với mã lệnh sau:

```
MsgBox "Bạn có muốn thoát khỏi chương trình không", vbCritical Or _
vbYesNo, "Thoát khỏi chương trình"
```



## 12.2. Hộp nhập dữ liệu (Input Box – InputBox)

InputBox được sử dụng nhằm yêu cầu người dùng nhập một chuỗi (String) theo gợi ý của dòng nhắc (Prompt) và tiêu đề (Title) trên đó. InputBox được gọi theo dạng hàm với giá trị trả về là chuỗi dữ liệu mà người dùng nhập.

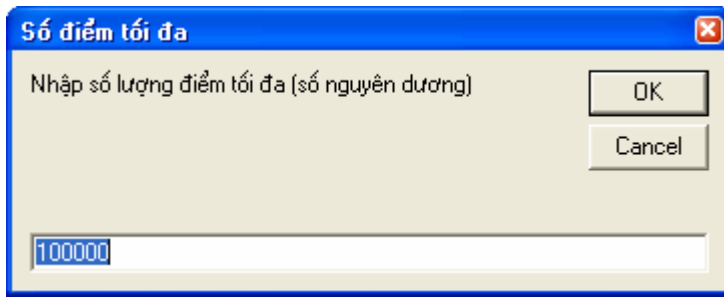
Cú pháp gọi InputBox thông thường như sau:

**InputBox(Prompt, [Title], [Default], [XPos], [YPos])**

Tham số	Mô tả
Prompt	Kiểu String. Nội dung dòng nhắc.
Title	Kiểu String. Nội dung tiêu đề.
Default	Kiểu Variant. Giá trị mặc định hiển thị trong InputBox.
XPos, YPos	Kiểu Double. Tọa độ góc trái trên của InputBox khi hiển thị ra màn hình.

Đoạn mã sau sẽ minh họa cách thức gọi InputBox:

```
Dim lngSodiumMax As Long
lngSodiumMax =
Val(InputBox("Nhập số lượng điểm tối đa (số nguyên dương)", _
"Số điểm tối đa", "100000"))
```



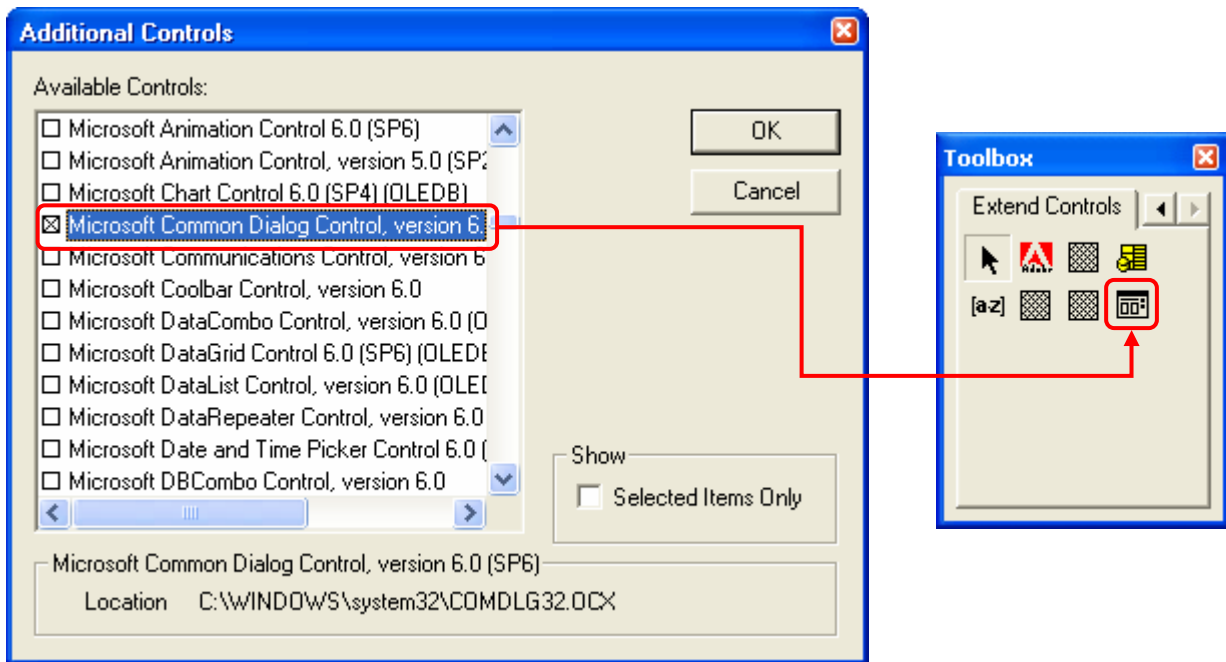
Nếu người dùng bấm nút OK thì giá trị trả về của hàm InputBox là một chuỗi có giá trị “100000”, còn nếu bấm nút Cancel thì giá trị trả về là một chuỗi rỗng.

### 12.3. Hộp thoại dựa trên điều khiển Common Dialog.

Điều khiển Common Dialog cho phép hiển thị các hộp thoại sau:

- ✦ Hộp thoại Open, Save: phục vụ thao tác mở và ghi tập tin một cách trực quan.
- ✦ Hộp thoại Color: phục vụ thao tác lựa chọn màu.
- ✦ Hộp thoại Font: phục vụ thao tác lựa chọn font chữ.
- ✦ Hộp thoại Print: phục vụ thao tác in ấn.

Để đưa điều khiển này vào trong hộp công cụ điều khiển (Control Toolbox) chọn menu Tools ⇒ Additional Controls, sau đó chọn Microsoft Common Dialog Control.



✦ Các thuộc tính của điều khiển Common Dialog.

Thuộc tính	Mô tả	Ghi chú
DialogTitle	Tiêu đề của hộp thoại	Kiểu String
FileName	Trả về đường dẫn và tên của file được chọn - Hộp thoại Open, Save.	Kiểu String
FileTitle	Trả về tên của file được chọn (không chứa đường dẫn) - Hộp thoại Open, Save.	Kiểu String
Filter	Mô tả các kiểu file sẽ được hiển thị trong hộp thoại - Hộp thoại Open, Save.	Kiểu String

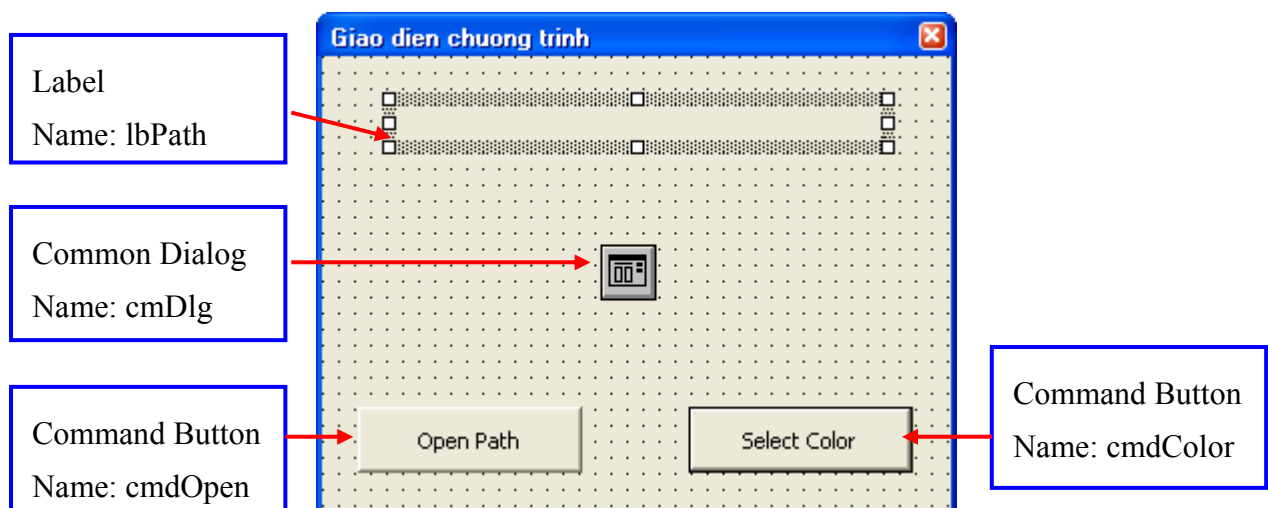
### CHƯƠNG III: CƠ BẢN VỀ NGÔN NGỮ LẬP TRÌNH VISUAL BASIC

FilterIndex	Kiểu file mặc định sẽ được hiển thị trong hộp thoại - Hộp thoại Open, Save.	Kiểu Integer
DefaultExt	Phần mở rộng mặc định được gán trong hộp thoại (khi người dùng không chọn mà nhập text vào trong phần tên file) - hộp thoại Open, Save.	Kiểu String
InitDir	Đường dẫn khởi tạo trong hộp thoại - Hộp thoại Open, Save	Kiểu String
CancelError	Qui định có phát sinh lỗi hay không khi người dùng chọn nút Cancel trong hộp thoại.	Kiểu Boolean
Color	Trả về màu được chọn trong hộp thoại – Hộp thoại Color	Tham khảo Object Browser

◆ Các phương thức của điều khiển Common Dialog.

Phương thức	Mô tả	Ghi chú
ShowOpen	Hiển thị hộp thoại mở file (Open)	Tham khảo trong Object Browser hoặc Help
ShowSave	Hiển thị hộp thoại ghi file (Save)	
ShowColor	Hiển thị hộp thoại chọn màu (Color)	
ShowFont	Hiển thị hộp thoại chọn font chữ (Font)	
ShowPrinter	Hiển thị hộp thoại in ấn (Printer)	

Ví dụ: Xây dựng UserForm gồm các điều khiển như hình dưới:



Yêu cầu:

- ◆ Người dùng bấm nút Open Path để lấy về đường dẫn của một file sau đó hiển thị nó trên điều khiển lbPath.
- ◆ Người dùng chọn nút Select Color để đổi màu của UserForm.

Mã lệnh tham khảo như sau:

Mã lệnh với thủ tục sự kiện Click của cmdOpen

```
Private Sub cmdOpen_Click()
    Dim strPath As String ' Xau lưu tru duong dan cua file duoc chon
    Dim strFilter As String ' Xau bieu dien cac kieu file hien thi
    strFilter = "App(*.exe)|*.exe|Text(*.txt)|*.txt|All files (*.*)|*.*"
    With cmDlg
        .DialogTitle = "Chon file"
        .InitDir = "C:\Program Files" ' duong dan mac dinh
    End With

```

```
.Filter = strFilter
.ShowOpen
strPath = .Filename ' lay ve ten day du cua file duoc chon
End With
lbPath.Caption = strPath
End Sub
```

Mã lệnh với thủ tục sự kiện Click của cmdColor

```
Private Sub cmdColor_Click()
    Dim lngColor As Long ' bien luu tru mau duoc chon
    With cmdDlg
        .ShowColor
        lngColor = .color ' lay ve mau nguoi dung chon trong hop thoai
    End With
    Me.BackColor = lngColor
End Sub
```

## 13. Lập trình xử lý tập tin

Xử lý tập tin là một nhu cầu không thể thiếu khi xây dựng phần mềm ứng dụng, bởi hầu hết các loại thông tin trên máy tính đều được lưu trữ trong các tập tin khác nhau. Trong các phần mềm ứng dụng đang được sử dụng, dữ liệu đầu vào của chúng được cung cấp dưới hai hình thức:

- ◆ Nhập trực tiếp từ bàn phím bởi người sử dụng: cách này chỉ phù hợp đối với lượng số liệu không nhiều.
- ◆ Nhập từ tập tin dữ liệu, ví dụ như để dựng được bản đồ số (để thiết kế đường ô tô trên máy tính) thì số liệu về các điểm đo toàn đặc khá nhiều (có thể là vài nghìn điểm đo) và thường được cung cấp dưới dạng các tập tin văn bản.

Việc nhập dữ liệu từ tập tin làm cho mức độ tự động hóa được nâng cao hơn, cho phép các phần mềm ứng dụng có thể kết nối được với nhau thông qua hình thức truyền dữ liệu. Ngoài ra, để lưu lại thông tin hay kết quả sau mỗi phiên làm việc với phần mềm ứng dụng, thì việc sử dụng tập tin làm nơi lưu trữ là phổ biến nhất. Các số liệu nhập vào cũng như các kết quả tính toán của phần mềm sẽ được lưu lại vào một hay nhiều tập tin và chúng sẽ được gọi lại trong phiên làm việc tiếp theo.

Để có thể xây dựng chương trình có khả năng nhập/xuất dữ liệu từ tập tin, thì người lập trình cần phải nắm được các nội dung sau:

- ◆ Kiểu của tập tin: là cách thức tổ chức dữ liệu trong tập tin đó. Hiện nay có vô số các định dạng cho tập tin bởi người dùng có thể tự do định nghĩa. Kiểu định dạng phổ biến nhất dùng để trao đổi dữ liệu là tập tin văn bản (thường có phần mở rộng là TXT, CSV). Với định dạng này ta có thể xem nội dung của tập tin bằng các chương trình soạn thảo đơn giản như Notepad.exe của Windows.
- ◆ Thao tác lên tập tin: là những thao tác nhằm biến đổi nội dung hoặc chính tập tin đó cho phù hợp với mục đích của người dùng. Những thao tác này được thực hiện theo một trình tự nhất định với các chương trình con chuyên trách cho từng nhiệm vụ. Các thao tác cơ bản bao gồm:
  - Đọc dữ liệu (Input) từ tập tin vào trong chương trình.
  - Ghi dữ liệu (Output) từ chương trình ra tập tin.
  - Tìm kiếm dữ liệu trong tập tin: đọc dữ liệu có chọn lọc.

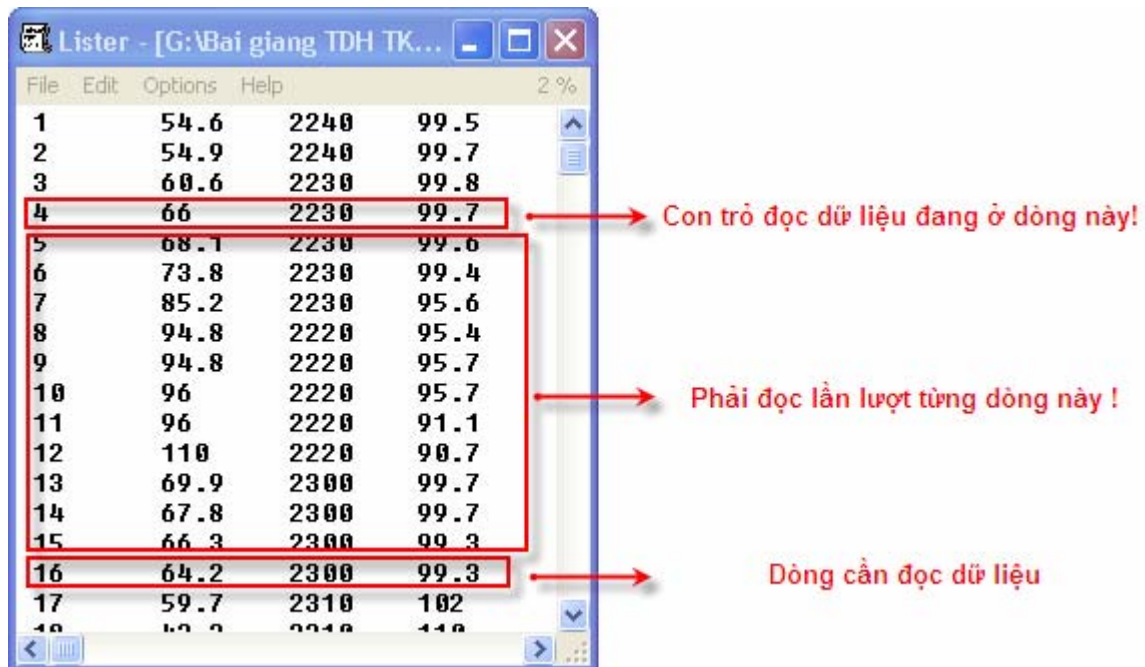


- Tạo mới tập tin: tạo ra tập tin trên đĩa để ghi dữ liệu lên nó.
- Xóa tập tin khi không còn dùng đến nó nữa.
- Di chuyển vị trí (Move) của tập tin từ nơi này đến nơi khác.
- Tạo bản sao (Copy) cho tập tin: tạo một tập tin thứ hai giống hệt tập tin gốc về nội dung nhưng tên hoặc vị trí lưu trữ của tập tin bản sao phải khác so với tập tin gốc.

### 13.1. Các hình thức truy cập tập tin

Truy cập tập tin bao gồm các thao tác đọc và ghi dữ liệu. Cách thức truy cập bao gồm:

- ◆ Truy cập kiểu tuần tự (Sequential): Quá trình đọc và ghi dữ liệu với tập tin theo các khối dữ liệu liên tục từ đầu đến cuối tập tin. Các khối dữ liệu liên tục có thể là các ký tự, các số, mẫu tin, chuỗi, dòng văn bản, ... Các khối này được phân cách nhau trong tập tin bằng ký tự dấu phẩy (,) hoặc ký tự xuống dòng. Ví dụ, muốn đọc dòng dữ liệu thứ n trong một tập tin văn bản có m dòng ( $m > n$ ), không thể ngay lập tức truy cập tới dòng thứ n mà phải lần lượt đọc từ dòng hiện tại (là dòng văn bản mà con trỏ đọc dữ liệu đang ở đó) tới dòng thứ n. Kiểu truy cập tuần tự thường áp dụng với các tập tin văn bản (text file).



Hình III-24: Truy cập kiểu tuần tự khi đọc tập tin văn bản.

- ◆ Truy cập kiểu ngẫu nhiên (Random): Quá trình đọc và ghi dữ liệu với tập tin được thực hiện dựa trên các mẫu tin có kích thước xác định (đơn vị để đo lường mẫu tin là Byte). Việc truy xuất đến một mẫu tin là tùy ý, không cần tuân theo trình tự mà theo thứ tự của mẫu tin đó trong tập tin. Quá trình truy cập ngẫu nhiên thường được áp dụng cho các tập tin trong đó dữ liệu được tổ chức theo các khối có cấu trúc (các mẫu tin).
- ◆ Truy cập kiểu nhị phân (Binary): Quá trình đọc và ghi dữ liệu với tập tin được thực hiện theo các khối không giống nhau về kích thước. Quá trình truy cập nhị phân thường áp dụng cho các tập tin có cấu trúc không cố định và dữ liệu có thể được xác định thông qua các byte dữ liệu được đọc vào.

Trong khuôn khổ giáo trình này, tập tin văn bản và các thao tác lên nó, được trình bày chi tiết bởi tính phổ biến và hữu dụng của loại tập tin này khi làm việc với các phần mềm ứng dụng trong ngành xây dựng công trình giao thông.

Để việc thao tác với các tập tin được thuận lợi, VB cung cấp sẵn hai phương pháp cơ bản:

- ✦ Sử dụng các hàm có sẵn để thao tác trực tiếp lên tập tin (dùng các hàm I/O).
- ✦ Sử dụng một số điều khiển để thao tác gián tiếp lên tập tin (dùng đối tượng FSO).

### 13.2. Xử lý dữ liệu trong tập tin với các hàm I/O:

Các hàm I/O (Input/Output) dùng để truy xuất các tập tin, trình tự như sau:

1. Mở tập tin: là yêu cầu bắt buộc phải thực hiện trước khi đọc hay ghi dữ liệu vào tập tin.
2. Thực hiện các thao tác với tập tin: đọc hoặc ghi dữ liệu vào tập tin.
3. Đóng tập tin: bắt buộc phải thực hiện khi kết thúc các thao tác với tập tin.

Trong khuôn khổ giáo trình này chỉ trình bày các thao tác theo kiểu tuần tự với tập tin, các kiểu truy cập khác có thể tìm trong các tài liệu tham khảo ghi ở cuối giáo trình này hoặc trong Help Online của VBA IDE.

#### 13.2.1. Mở tập tin:

Cú pháp:

```
Open <đường dẫn> For [Kiểu thao tác] as <filename> [Len=Buffersize]
```

Trong đó:

- ✦ <đường dẫn>: là một giá trị kiểu String dùng để xác định đường dẫn của tập tin (vị trí của nó trên đĩa).
- ✦ <Kiểu thao tác>: cách thức thao tác với tập tin, tham số này có thể nhận một trong các giá trị sau:
  - Input: đọc dữ liệu từ tập tin, để không gây lỗi thì tập tin này phải có sẵn trên đĩa.
  - Output: ghi dữ liệu vào tập tin với hai điểm cần lưu ý:
    - ◆ Nếu tập tin là có sẵn thì toàn bộ dữ liệu bên trong nó sẽ bị xóa sạch trước khi dữ liệu mới được ghi vào (ghi đè lên những dữ liệu đã có). Việc này sẽ làm mất đi những dữ liệu ban đầu.
    - ◆ Nếu tập tin chưa tồn tại, một tập tin mới sẽ được tạo ra với tên và vị trí của tập tin được xác định trong <đường dẫn>.
    - ◆ Append: ghi dữ liệu vào cuối tập tin đã có (ghi thêm, nối vào những dữ liệu đã có).
  - <filename>: là một giá trị kiểu Integer đại diện cho tập tin đó. Sau này, khi thao tác với tập tin này, thì giá trị này sẽ là đại diện. Điều này rất hữu ích khi làm việc đồng thời với nhiều tập tin đang mở, lúc đó, để ghi hay đọc dữ liệu với tập tin nào, ta chỉ việc đưa vào giá trị của <filename> tương ứng trong các lệnh đọc/ghi dữ liệu.
  - [Len = Buffersize]: chỉ ra số ký tự trong vùng đệm khi sao chép dữ liệu giữa tập tin và chương trình. Đây là một giá trị tùy chọn.

Ví dụ: Khi trên đĩa C không có tập tin File1.txt thì câu lệnh sau sẽ tạo mới và mở sẵn tập tin này để ghi dữ liệu:

```
Open "C:\file1.txt" For Output as 1
```

#### 13.2.2. Đọc dữ liệu từ tập tin:

Sau khi tập tin đã được mở bằng lệnh Open với kiểu là Input, nó đã sẵn sàng cho việc đọc dữ liệu bên trong nó. Dữ liệu có thể đọc theo những cách thức sau:

### Đọc dữ liệu theo từng dòng

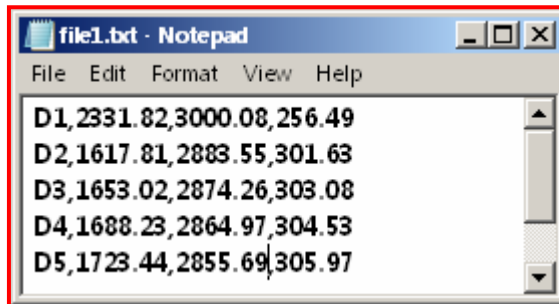
Khái niệm dòng dữ liệu trong tập tin khác so với khái niệm dòng chữ trên trang giấy. Dòng dữ liệu có thể chứa rất nhiều ký tự (có độ dài hầu như không hạn chế) và một dòng được coi là kết thúc tại nơi có chứa ký hiệu xuống dòng (vbCrLf – bao gồm hai ký tự có số hiệu 13 và 10). Cú pháp đọc một dòng từ tập tin như sau:

```
Line Input #<filename>, <strVar>
```

Câu lệnh này đọc dữ liệu từ dòng hiện tại của tập tin đã được mở (có chỉ số là <filename>) và gán dữ liệu đọc được cho biến strVar (biến này có kiểu String). Câu lệnh Line Input # sẽ tự động nhận dạng dòng dữ liệu thông qua ký hiệu xuống dòng (tuy nhiên nó không đưa ký hiệu xuống dòng vào biến strVar). Sau lệnh Line Input #, vị trí con trỏ đọc dữ liệu sẽ được tự động chuyển xuống dòng tiếp theo.

**CHÚ Ý** Ngay khi mở tập tin để đọc, con trỏ đọc dữ liệu sẽ được tự động đặt ở dòng đầu tiên trong tập tin.

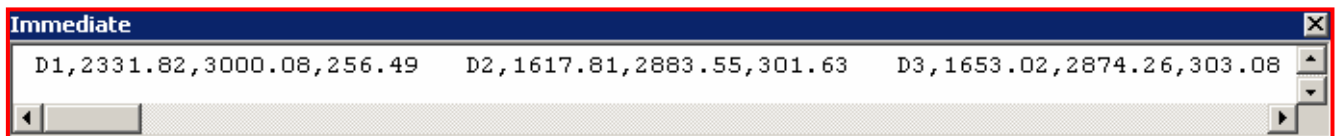
Ví dụ: Một tập tin văn bản có đường dẫn “C:\file1.txt” với nội dung như sau:



Mã lệnh sau sẽ đọc nội dung của 3 dòng dữ liệu đầu tiên trong tập tin:

```
Dim strRe1 As String, strRe2 As String, strRe3 As String
Open "C:\file1.txt" For Input As 1
Line Input #1, strRe1
Line Input #1, strRe2
Line Input #1, strRe3
Debug.Print strRe1, strRe2, strRe3
Close 1
```

Kết quả thực hiện của đoạn mã lệnh trên như sau:



**CHÚ Ý** Khi kết thúc thao tác với tập tin thì cần phải đóng chúng lại, nếu không thông tin trong đó có thể mất hoặc người khác không truy cập vào tập tin đó được.

### Đọc một danh sách các chuỗi theo ký tự phân cách

Đọc một danh sách các chuỗi theo ký tự phân cách là dấu phẩy (,) hoặc ký hiệu xuống dòng (vbCrLf) với cú pháp sau:

**Input # <filenumber>, <danh sách các biến>**

Câu lệnh này đọc khối dữ liệu từ vị trí hiện tại của con trỏ đọc dữ liệu trong tập tin có chỉ số <filenumber>. Dữ liệu đọc được sẽ được gán vào cho <danh sách các biến> (mỗi biến trong danh sách này đều có kiểu dữ liệu là String). Số khối dữ liệu được đọc sẽ phụ thuộc vào số biến có trong <danh sách các biến>. Khối dữ liệu được nhận dạng dựa vào dấu phẩy (,) hoặc ký hiệu xuống dòng (vbCrLf). Sau lệnh Input #n, vị trí con trỏ đọc dữ liệu sẽ được tự động chuyển sang khối dữ liệu tiếp theo.

**CHÚ Ý** Đọc dữ liệu bằng lệnh Input #n thường được dùng với tập tin mà dữ liệu của nó được tạo ra bởi lệnh Write #n.

Ví dụ: Với tập tin văn bản "C:\file1.txt" như trên, với các mã lệnh sau:

```
Dim strRe1 As String, strRe2 As String, strRe3 As String
Open "C:\file1.txt" For Input As 1
Input #1, strRe1, strRe2, strRe3
Debug.Print strRe1, strRe2, strRe3
Close 1
```

Ta nhận được kết quả như hình dưới:



Nếu lệnh đọc dữ liệu được gọi khi vị trí con trỏ đọc dữ liệu ở cuối tập tin thì sẽ xảy ra lỗi. Để tránh lỗi này cần phải kiểm tra vị trí của con trỏ đọc dữ liệu, xem nó có ở cuối tập tin hay không. Hàm EOF (FileNumber) (có kiểu Boolean) được dùng cho mục đích này, nó sẽ trả về giá trị True nếu vị trí con trỏ đọc dữ liệu đang ở cuối tập tin, và ngược lại sẽ trả về giá trị False.

Ví dụ sau sẽ đọc toàn bộ dữ liệu trong tập tin C:\File1.txt:

```
Dim strRe As String
Open "C:\file1.txt" For Input As #1
Do While Not EOF(1)
    Input #1, strRe
    Debug.Print strRe
Loop
Close #1
```

### 13.2.3. Ghi dữ liệu vào tập tin:

Thao tác ghi dữ liệu vào tập tin được thực hiện sau khi tập tin đã mở để ghi với hai kiểu ghi dữ liệu là ghi đè lên dữ liệu ban đầu (với thông số Output) hay ghi nối vào sau các dữ liệu ban đầu (với thông số Append). Với Output: toàn bộ nội dung ban đầu của tập tin sẽ bị xóa và con trỏ ghi dữ liệu sẽ được đặt ở vị trí đầu tiên. Nếu tập tin chưa có thì nó sẽ được tự động tạo ra theo tên và vị trí của đường dẫn trong lệnh Open. Với Append: việc ghi được thực hiện nối tiếp vào tập tin hiện tại, vị trí bắt đầu ghi mặc định là cuối tập tin.

### Ghi dữ liệu với lệnh Print #n

Cú pháp như sau:

**Print # <filename>, [outputlist]**

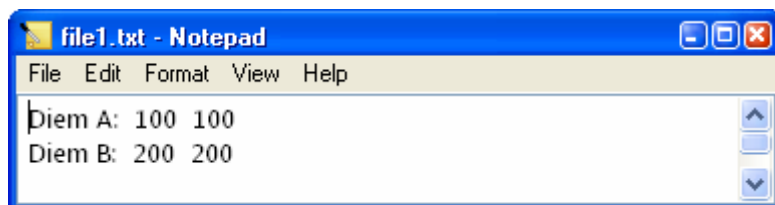
Trong đó:

- ◆ filename: chỉ số của tập tin.
- ◆ outputlist: danh sách các giá trị cần ghi, các giá trị trong danh sách này được phân tách nhau bởi dấu ( ; ). Nếu outputlist kết thúc bằng dấu (;) con trỏ ghi dữ liệu sẽ chuyển sang vị trí kế tiếp. Ngược lại, nếu cuối danh sách để trống thì con trỏ ghi dữ liệu sẽ chuyển sang dòng kế tiếp. Các thành phần dữ liệu trong outputlist sẽ được ghi liên tục vào tập tin, người dùng có thể thêm các khoảng trống bằng lệnh Spc (n) hoặc các dấu tab bằng lệnh Tab (n) (với n là số ký tự cần thêm vào).

Ví dụ: chương trình sau sẽ ghi dữ liệu vào tập tin "C:\file1.txt" bằng lệnh Print #

```
Sub FilePrint()
    Open "C:\file1.txt" For Output As 1
    Dim Ax As Double, Ay As Double
    Dim Bx As Double, By As Double
    Ax = 100: Ay = 100
    Bx = 200: By = 200
    Print #1, "Diem A: "; Ax;
    Print #1, Ay
    Print #1, "Diem B: "; Bx;
    Print #1, By
    Close 1
End Sub
```

Kết quả như sau:



**Ghi dữ liệu với lệnh write #**

Cú pháp như sau:

**Write #filename, [outputlist]**

Trong đó:

- ◆ filename: chỉ số của tập tin.
- ◆ outputlist: danh sách các giá trị cần ghi, các giá trị trong danh sách được phân tách nhau bởi dấu ( , ). Nếu outputlist kết thúc bằng dấu ( ; ) con trỏ ghi dữ liệu sẽ chuyển sang vị trí kế tiếp. Ngược lại, nếu cuối danh sách để trống thì con trỏ ghi dữ liệu sẽ chuyển sang dòng kế tiếp. Các thành phần dữ liệu trong outputlist sẽ được ghi liên tục vào tập tin và dấu phẩy ( , ) sẽ được tự động thêm vào giữa hai giá trị trong tập tin.

Ví dụ: chương trình con sau sẽ ghi dữ liệu vào tập tin "C:\file2.txt":

```
Sub FileWrite()
    Open "C:\file2.txt" For Output As 1
    Dim Ax As Double, Ay As Double
```

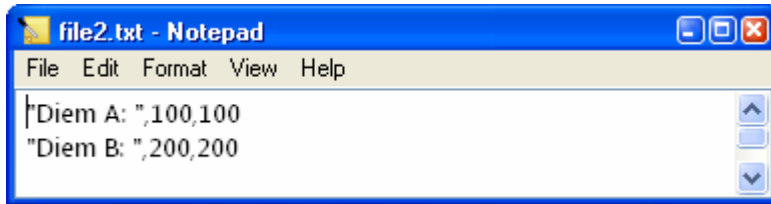
```
Dim Bx As Double, By As Double  
Ax = 100: Ay = 100  
Bx = 200: By = 200
```

```
Write #1, "Diem A: ", Ax;
```

```
Write #1, Ay  
Write #1, "Diem B: ", Bx;  
Write #1, By  
Close 1
```

```
End Sub
```

Kết quả như sau:



### 13.2.4. Đóng tập tin

Sau khi thao tác đọc/ghi dữ liệu lên tập tin ta cần phải đóng chúng lại bằng lệnh `Close` theo cú pháp sau:

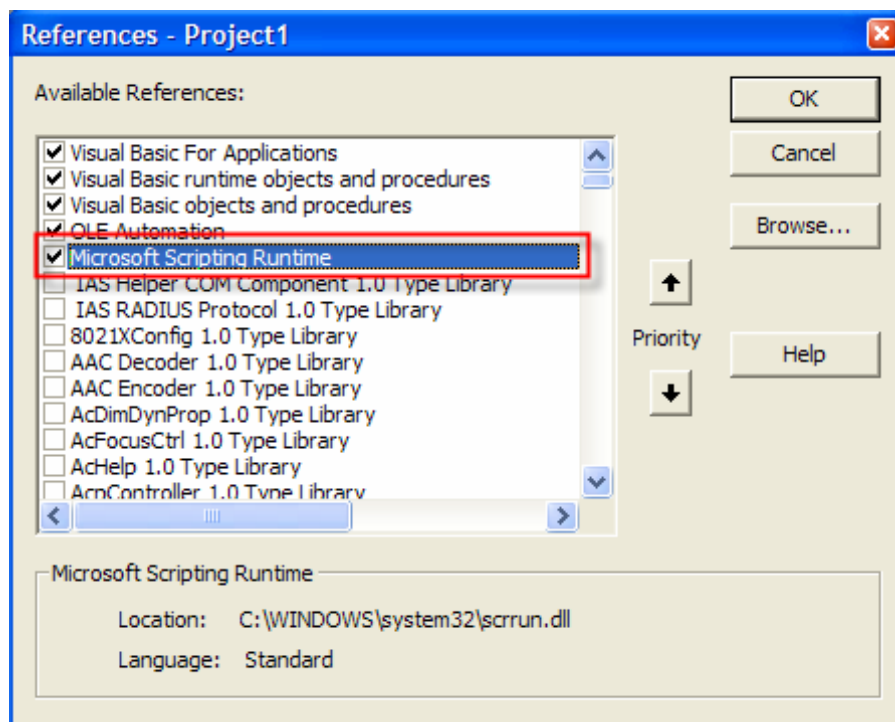
```
Close <filename>
```

**CHÚ Ý** Trong tất cả các ví dụ đọc và ghi dữ liệu trên đều có lệnh đóng tập tin sau khi kết thúc các thao tác đọc/ghi.

## 13.3. Xử lý dữ liệu trong tập tin theo mô hình FSO (File System Object)

Các thao tác với tập tin ở phần trên chỉ bao gồm hai loại cơ bản nhất là đọc dữ liệu từ tập tin và ghi thông tin lên tập tin, còn những thao tác khác, thường xuyên được sử dụng, như: lựa chọn tập tin, sao chép, di chuyển, xóa..., tuy có thể thực hiện được từ những lệnh đọc/ghi cơ bản trên nhưng khá rắc rối. Vì vậy, để tạo thuận lợi cho người dùng, VB đã cung cấp những chức năng này thông qua mô hình FSO. Đây là một tập hợp các lớp đối tượng, mà nhiệm vụ của chúng là cung cấp cho người dùng hầu hết các công cụ thao tác với tập tin.

Các lớp đối tượng theo mô hình FSO là một dạng bổ sung cho VBA và được cung cấp dưới dạng thư viện lập trình với tên gọi "Microsoft Scripting Runtime". Để sử dụng thư viện này trong VBA IDE cần thực hiện thao tác sau: trong VBAIDE chọn trình đơn **Tools** ⇒ **References** ⇒ Đánh dấu chọn **Microsoft Scripting Runtime** ⇒ Chọn **OK**



Hình III-25: Sử dụng thư viện lập trình Microsoft Scripting Runtime.

**CHÚ Ý** Khi thao tác với tập tin, mô hình FSO chỉ hỗ trợ cách thức truy cập tuần tự.

✦ Các lớp (class) chính trong mô hình FSO:

Tên lớp	Mô tả	Ghi chú
FileSystemObject	Đối tượng quản lý trong mô hình FSO	Tham khảo trong Object Browser hoặc Help.
Drive	Đối tượng ổ đĩa	
Folder	Đối tượng thư mục	
File	Đối tượng tập tin	
TextStream	Đối tượng luồng dữ liệu (dạng text) phục vụ việc thao tác với dữ liệu trong tập tin	

✦ Các phương thức chính của lớp FileSystemObject phục vụ cho thao tác tập tin

Tên phương thức	Mô tả	Ghi chú
CopyFile	Sao chép tập tin	
DeleteFile	Xoá tập tin	Tham khảo Object Browser
MoveFile	Di chuyển tập tin	Tham khảo Object Browser
FileExists	Kiểm tra sự làm việc của tập tin	Trả về giá trị Boolean
CreateTextFile	Tạo tập tin mới (dạng text)	Trả về đối tượng kiểu TextStream
GetFile	Nhận về một tập tin đã có	Trả về đối tượng kiểu File
OpenTextFile	Mở một tập tin dạng text để làm việc	Trả về đối tượng kiểu TextStream

✦ Các phương thức của lớp TextStream

Tên phương thức	Mô tả	Ghi chú
Read	Đọc một xâu dữ liệu trong tập tin	Trả về dữ liệu kiểu String

ReadLine	Đọc một dòng dữ liệu trong tập tin	Trả về dữ liệu kiểu String
ReadAll	Đọc toàn bộ dữ liệu trong tập tin	Trả về dữ liệu kiểu String
Skip	Bỏ qua một xâu dữ liệu trong tập tin	Trả về đối tượng kiểu TextStream
SkipLine	Bỏ qua một dòng dữ liệu trong tập tin	
Write	Ghi một xâu dữ liệu vào trong tập tin	
WriteLine	Ghi một xâu dữ liệu thành một dòng trong tập tin	
WriteBlankLines	Chèn một dòng trống vào trong tập tin	
Close	Đóng luồng dữ liệu.	

### Trình tự làm việc với dữ liệu của tập tin theo mô hình FSO

1. Tạo đối tượng <FSO> thuộc lớp `FileSystemObject` nhằm quản lý tập tin, thư mục hoặc ổ đĩa theo cú pháp sau:

```
Dim FSO As New FileSystemObject
```

Hoặc:

```
Set FSO = CreateObject("Scripting.FileSystemObject")
```

Trong đó: <FSO> là tên của đối tượng (chính là tên biến), mà dựa vào nó ta sẽ thao tác với tập tin.

2. Tạo đối tượng <TxtStr> thuộc lớp `TextStream` nhằm phục vụ cho việc thao tác với dữ liệu trong tập tin theo cú pháp sau:

```
Dim TxtStr As New TextStream
```

3. Thao tác với dữ liệu với đối tượng `TxtStr`.
4. Đóng luồng dữ liệu để kết thúc thao tác theo cú pháp sau

```
TxtStr.Close
```

---

**CHÚ Ý** Hai cách khai báo biến đối tượng sau là tương đương nhau:

```
Dim FSO As New FileSystemObject
```

Và:

```
Dim FSO As FileSystemObject
```

```
Set FSO = New FileSystemObject
```

---

#### 13.3.1. Tạo tập tin mới

Sử dụng phương thức `CreateTextFile` để tạo tập tin mới và mở sẵn nó cho các thao tác đọc/ghi. Cú pháp như sau:

```
Set TxtStr=FSO.CreateTextFile(FileName, [Overwrite], [Unicode])
```

Trong đó:



- ◆ **FileName**: tên của tập tin cần tạo, kiểu `String`, phải bao gồm đầy đủ đường dẫn để xác định vị trí của tập tin, nếu chỉ có tên tập tin thì tập tin này sẽ được tạo vào thư mục mặc định.
- ◆ **[Overwrite]**: lựa chọn có ghi đè hay không trong trường hợp tập tin đã có. Tham số này là tùy chọn và có kiểu là `Boolean`, giá trị mặc định là `True` (cho phép ghi đè). Nếu đặt tham số này là `False` và tập tin đã có thì sẽ phát sinh lỗi và làm dừng chương trình.
- ◆ **[Unicode]**: lựa chọn có sử dụng bảng mã `Unicode` trong tập tin hay không. Tham số này là tùy chọn và có kiểu là `Boolean`, giá trị mặc định là `False`.

Ví dụ sau sẽ tạo ra tập tin `Test.txt` trong ổ đĩa `C`, nếu tập tin này đã có, nó sẽ bị ghi đè lên, nghĩa là các thông tin cũ sẽ bị xóa hết:

```
Dim FSO As New FileSystemObject
Dim TxtStr As TextStream
Set TxtStr=FSO.CreateTextFile("C:\Test.txt", True, True)
```

### 13.3.2. Mở tập tin đã có để thao tác

Khi muốn làm việc với một tập tin đã có (đọc/ghi), sử dụng cú pháp sau:

```
Set TxtStrObj=FSO.OpenTextFile(FileName, [IOMode], [Create], [Format])
```

Trong đó:

- ◆ **FileName**: Tên và vị trí của tập tin (kiểu `String`).
- ◆ **[IOMode]**: Kiểu thao tác với tập tin. Tham số này là tùy chọn, có thể nhận một trong 3 giá trị sau:
  - ◆ `ForAppending` (hoặc 8): thêm dữ liệu vào cuối tập tin đã có.
  - ◆ `ForReading` (hoặc 1): đọc dữ liệu từ tập tin. Đây là giá trị mặc định của tham số.
  - ◆ `ForWriting` (hoặc 2): ghi dữ liệu vào tập tin.
- ◆ **[Create]**: Tùy chọn có tạo tập tin hay không trong trường hợp tập tin chưa tồn tại. Nó có kiểu là `Boolean`, giá trị mặc định là `False`.
- ◆ **[Format]**: tham số tùy chọn, chỉ cách mở tập tin theo định dạng. Tham số này có thể nhận một trong 3 giá trị sau:
  - ◆ `TristateUseDefault` (hoặc -2): mở tập tin theo định dạng chuẩn của hệ thống.
  - ◆ `TristateTrue` (hoặc -1): mở tập tin với định dạng `Unicode`.
  - ◆ `TristateFalse` (hoặc 0): mở tập tin với định dạng theo chuẩn `ASCII`. Đây là giá trị mặc định của tham số.

Trong quá trình đọc dữ liệu từ tập tin, phải luôn chắc chắn rằng vị trí con trỏ đọc dữ liệu không ở cuối tập tin bởi điều này sẽ làm phát sinh lỗi. Để kiểm tra xem vị trí con trỏ đọc dữ liệu đã ở cuối tập tin chưa, dùng thuộc tính `AtEndOfStream` của lớp `TextStream`. Thuộc tính này trả về giá trị `True` nếu ở cuối, trả về `false` nếu chưa.

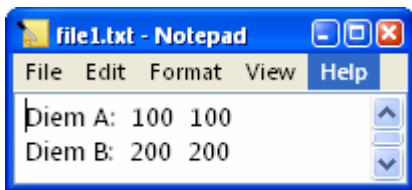
Ví dụ: đoạn chương trình sau sẽ đọc nội dung của tập tin `"C:\file1.txt"` và in ra cửa sổ `Immediate`.

```
Sub FSOReadFile()
    Dim FSO As New FileSystemObject
```

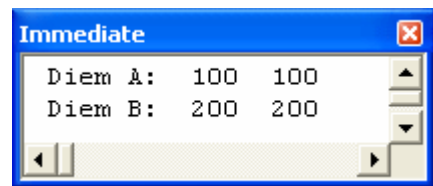
```
Dim TxtStr As TextStream
Dim StrTemp As String

If FSO.FileExists("C:\file1.txt") Then
    Set TxtStr = FSO.OpenTextFile("C:\file1.txt", ForReading)
    Do While Not (TxtStr.AtEndOfStream)
        StrTemp = TxtStr.ReadLine
        Debug.Print StrTemp
    Loop
    TxtStr.Close
Else
    MsgBox "Tap tin không co hoac Duong dan sai",vbCritical,"Thong bao"
End If
End Sub
```

Kết quả thực thi đoạn chương trình trên như sau:



**Tập tin**



**Kết quả**

Những thao tác khác như Copy, Move, Delete hay làm việc với thư mục không được đề cập trong giáo trình này, tuy nhiên người đọc có thể tìm hiểu trong các tài liệu tham khảo nêu ở cuối giáo trình này hoặc trong Help Online của VBA IDE.

## 14. Gỡ rối và bắt lỗi trong VBAIDE

Trong quá trình xây dựng một dự án phần mềm, việc gặp các lỗi là không thể tránh khỏi. Vì vậy, việc tìm và xử lý lỗi là điều tất yếu. Trình tự của công việc này như sau:

1. Tìm và phân loại lỗi.
2. Tìm kiếm vị trí mã lệnh phát sinh lỗi.
3. Sửa lỗi.
4. Ngăn chặn lỗi có thể xảy ra trong tương lai (bắt lỗi).

### 14.1. Phân loại lỗi trong lập trình

Các lỗi có thể được phân loại như sau:

- ✦ Lỗi cú pháp (Syntax Error): là các lỗi phát sinh do viết mã lệnh sai quy tắc. Ví dụ: đặt tên biến trùng từ khoá, viết sai từ khoá,... Tuy nhiên trong VBA IDE, các lỗi cú pháp được hạn chế rất nhiều nhờ các tính năng phát sinh mã lệnh tự động, gợi ý mã lệnh hoặc tự động kiểm tra cú pháp của mã lệnh. Một chương trình chỉ chạy khi không còn lỗi cú pháp.
- ✦ Lỗi khi chạy chương trình (Runtime Error): là các lỗi phát sinh trong khi chương trình đang chạy. Đây là một loại lỗi mà nguyên nhân gây lỗi rất đa dạng cho nên việc phát hiện và sửa chữa lỗi loại này khá khó khăn. Ví dụ như lỗi do tràn bộ nhớ, các tài nguyên mà chương trình cần sử dụng không có trong hệ thống,... Các lỗi thực thi thường dẫn tới sự chấm dứt hoạt động của chương trình, thậm chí của toàn bộ hệ thống.

- ◆ **Lỗi do giải thuật:** là các lỗi xảy ra do thuật toán hoặc do việc cài đặt và sử dụng các thuật toán chưa đúng. Các lỗi giải thuật thường dẫn tới kết quả xử lý của chương trình bị sai, trong nhiều trường hợp các lỗi giải thuật cũng có thể là nguyên nhân làm phát sinh các lỗi thực thi. VBA IDE không thể phát hiện được các lỗi loại này mà phải do người lập trình hoặc người sử dụng chương trình mới tìm ra được. Vì vậy, đây là loại lỗi khó phát hiện và khắc phục nhất.

## 14.2. Gỡ rối trong lập trình

Các lỗi cú pháp có thể được khắc phục khá dễ dàng do người lập trình được thông báo của trình biên dịch ngay trong quá trình viết mã lệnh (tham khảo thêm mục “*Các trợ giúp về cú pháp trong quá trình viết mã lệnh*” trang 25 và “*Tính năng gợi nhớ và tự hoàn thiện mã lệnh*” trang 26). Ngoài ra, cũng có một số lỗi về cú pháp mà VBA IDE không thể phát hiện ngay lúc viết mã lệnh được, với những trường hợp này, thông thường ngay trước khi chương trình được thực thi, VBA IDE sẽ báo lỗi với người dùng.

Do các lỗi cú pháp rất dễ dàng được phát hiện nên phần này sẽ tập trung vào các tính năng dùng để phát hiện lỗi thực thi và lỗi giải thuật.

### 14.2.1. Phát hiện lỗi lúc thực thi

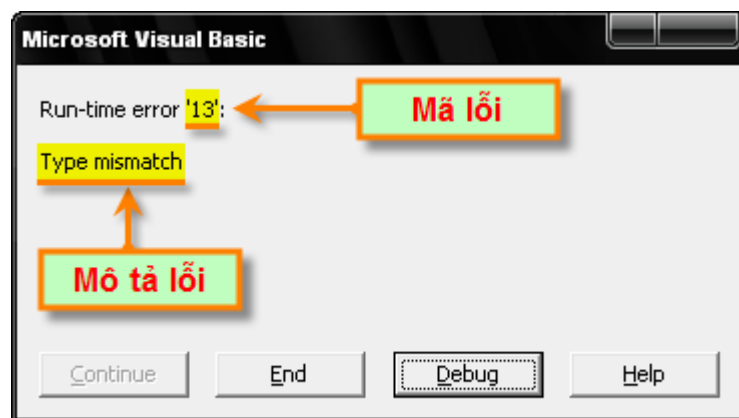
Đối với các lỗi phát sinh lúc thực thi chương trình, VBA IDE sẽ tự động dừng chương trình và hiển thị thông báo lỗi, sau đó cho phép người dùng lựa chọn kết thúc chương trình hoặc tiếp tục gỡ rối chương trình.

Để hiểu rõ hơn tính năng này, nhập đoạn mã lệnh sau vào mô-đun chuẩn của VBA IDE

```
Sub VDLoiThucThi ()
    Dim i As Integer
    i = InputBox("Nhập số nguyên: ", "VD loi thuc thi")
    MsgBox i
End Sub
```

Đoạn mã lệnh trên sẽ hiển thị hộp thoại `InputBox` để người dùng nhập một số nguyên, sau đó hiển thị kết quả vừa được nhập vào thông qua hàm `MsgBox`.

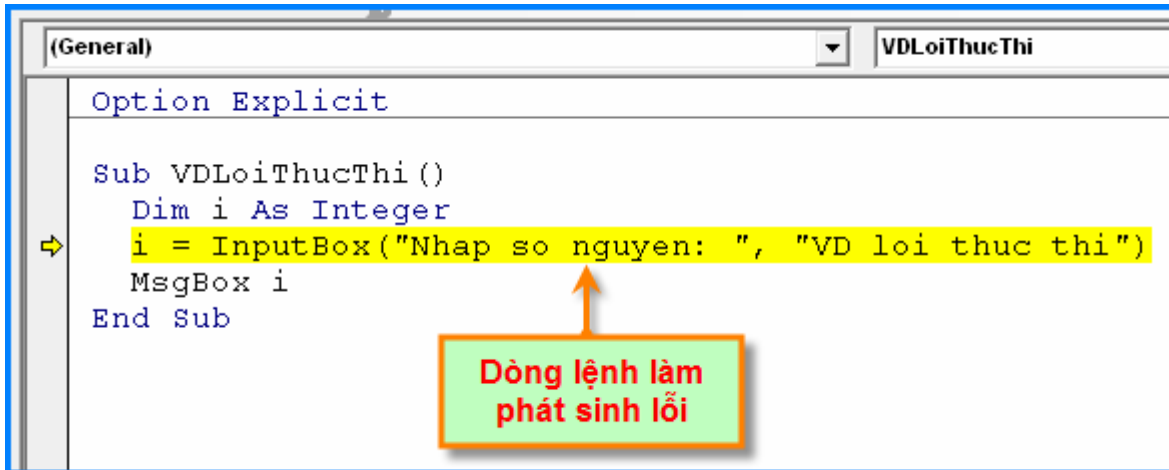
Thực thi chương trình con này, sau đó trong hộp thoại vừa hiển thị, nhập vào một chuỗi ký tự là số nguyên, ví dụ là 123, sau đó nhấn OK ⇒ một hộp thoại khác sẽ hiển thị kết quả vừa nhập. Tiếp tục thực thi chương trình một lần nữa, lần này nhập một chuỗi ký tự không phải là số nguyên, ví dụ là “ABC”, sau đó chọn OK. VBA IDE sẽ hiển thị thông báo lỗi như sau:



Hình III-26: Thông báo lỗi phát sinh lúc thực thi chương trình.

Trong cửa sổ thông báo lỗi có hiển thị Mã lỗi và Mô tả lỗi để người dùng có thể tra cứu, khắc phục lỗi. Trong ví dụ này, đó là lỗi số 13, lỗi “Type mismatch – Không phù hợp kiểu dữ liệu”. Nếu người dùng chọn nút lệnh **End** ⇨ chương trình sẽ kết thúc thực thi.

Nếu người dùng chọn nút lệnh **Debug**, chương trình sẽ dừng lại ngay tại dòng lệnh đã làm phát sinh lỗi trên. VBA IDE sẽ hiển thị cửa sổ mã lệnh và đánh dấu dòng lệnh nơi phát sinh ra lỗi thực thi.



**Hình III-27: VBA IDE đánh dấu dòng lệnh làm phát sinh lỗi thực thi.**

Nhờ có điều này mà người lập trình có thể rõ được nguyên nhân phát sinh lỗi và nơi làm phát sinh lỗi thực thi, để từ đó có được hướng khắc phục hợp lý.

### **14.2.2. Các phương pháp thực thi mã lệnh**

Trong các trình biên dịch hiện đại nói chung và VBAIDE nói riêng, người dùng được hỗ trợ rất nhiều thông qua các tính năng gỡ rối như biên dịch theo từng bước, theo các điểm dừng, hiển thị các kết quả trung gian. Tùy thuộc vào mục đích mà người lập trình có thể sử dụng một phương pháp phù hợp hoặc có thể sử dụng phối hợp giữa các phương pháp. Các phương pháp thực thi mã lệnh có thể được truy cập thông qua trình đơn Debug của VBAIDE:

#### **Chạy từng bước (Step Into)**

Nút lệnh: . Phím tắt: **F8**.

Chương trình được dịch theo từng dòng lệnh. Mỗi khi người lập trình nhấn F8 thì chương trình sẽ thực thi một dòng lệnh, cứ như thế cho đến khi kết thúc chương trình.


Nếu tại một dòng lệnh có lời gọi đến chương trình con khác thì khi tiếp tục thực hiện với Step Into, con trỏ biên dịch sẽ được nhảy đến dòng đầu tiên của chương trình con được gọi.

#### **Chạy từng bước với khối lệnh (Step Over)**

Nút lệnh: . Phím tắt: **SHIFT+F8**.

Phương pháp này tương tự như chạy từng bước (Step Into) nhưng việc thực thi một chương trình con được coi như thực thi một dòng lệnh. Vì vậy nếu trong chương trình hiện tại có một lời gọi chương trình con thì chương trình con sẽ được thực thi như một lệnh và do đó con trỏ biên dịch sau đó sẽ nhảy tới dòng lệnh tiếp theo của chương trình con hiện tại.

#### **Chạy ra ngoài chương trình con (Step Out)**

Nút lệnh: . Phím tắt: **CTRL+SHIFT+F8**.

Nếu con trỏ biên dịch đang ở trong một chương trình con, thì lệnh biên dịch Step Out sẽ dịch toàn bộ các lệnh còn lại trong chương trình con đó và đưa con trỏ lệnh tới vị trí tiếp sau vị trí có lời gọi chương trình con.

### Chạy tới vị trí con trỏ chuột (Run to Cursor)

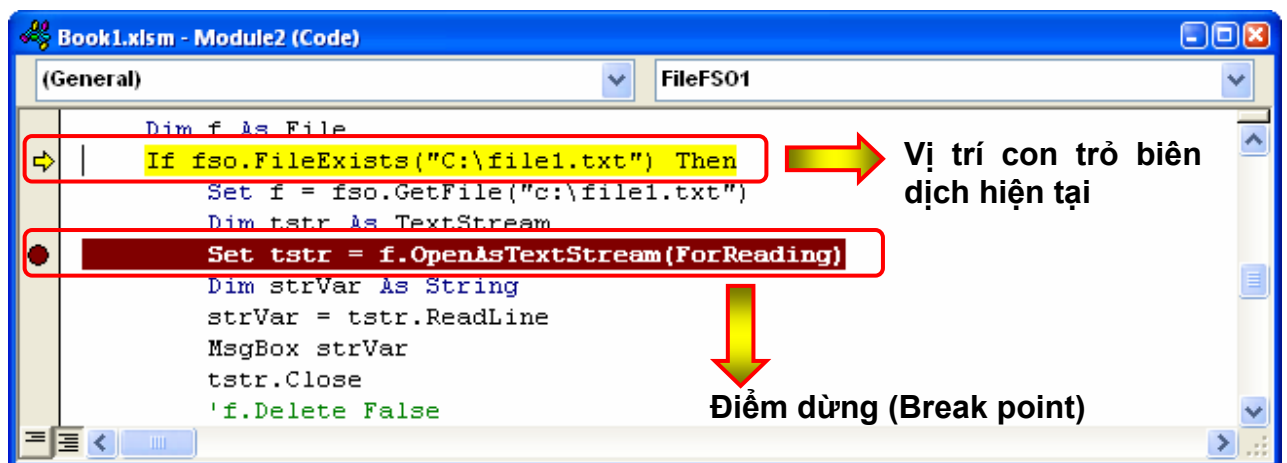
Nút lệnh: . Phím tắt: **CTRL+F8**.

Chạy từ vị trí con trỏ biên dịch hiện tại tới vị trí có con trỏ soạn thảo. Phương pháp này thường được dùng khi người lập trình muốn thực thi qua toàn bộ những khối lệnh lập đến dòng lệnh mà mình cần quan tâm.

### Tạo điểm dừng (Break point) khi chạy chương trình

Nút lệnh: . Phím tắt: **F9**.

Với phương pháp này, khi người lập trình thực thi chương trình, trình biên dịch sẽ dừng lại tại các vị trí dòng lệnh tương ứng đã được đánh dấu trước. Để tạo điểm dừng cho một dòng lệnh, đưa con trỏ soạn thảo chọn dòng lệnh tương ứng và nhấn phím **F9**.



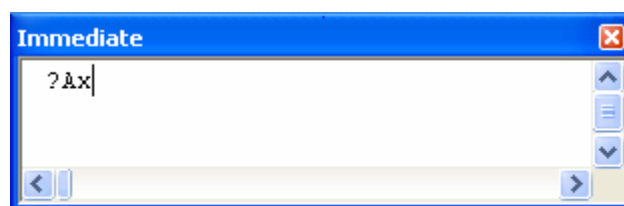
Nếu muốn xóa điểm dừng cho một dòng lệnh, đưa con trỏ soạn thảo đến dòng lệnh đó có điểm dừng và nhấn phím **F9**. Nếu muốn xóa hết tất cả các điểm dừng đã tạo, nhấn phím tắt **CTRL+SHIFT+F9**.

#### 14.2.3. Cửa sổ trợ giúp gỡ rối

Ngoài việc gỡ rối sử dụng các phương pháp thực thi chương trình, VBAIDE còn hỗ trợ người lập trình các công cụ dùng để thử nghiệm các dòng lệnh và kiểm soát các biến trong chương trình. Đây là công cụ rất hữu ích giúp người lập trình có thể theo dõi và từ đó phát hiện ra lỗi trong chương trình, nhất là các lỗi phát sinh do giải thuật.

### Cửa sổ trung gian (Immediate Window).

Để hiển thị cửa sổ trung gian, trong VBAIDE chọn trình đơn **View** ⇒ **Immediate window**, hoặc sử dụng phím tắt **CTRL+G**:



Hình III-28: Cửa sổ trung gian.

Với cửa sổ trung gian, người dùng có thể:

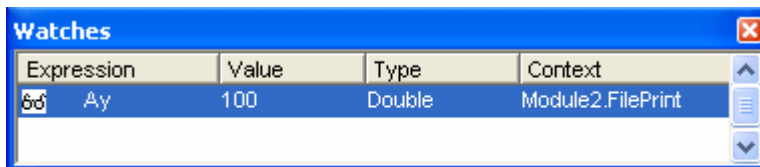
- ✦ Gõ một dòng lệnh vào và nhấn ENTER để thực thi dòng lệnh đó trực tiếp từ cửa sổ trung gian.
- ✦ Hiển thị giá trị của biểu thức lên cửa sổ trong quá trình gỡ rối. Để hiển thị giá trị của biểu thức, trong cửa sổ trung gian gõ "?Biểu\_Thức" rồi nhấn phím ENTER.
- ✦ Người lập trình có thể in giá trị của biểu thức ra cửa sổ trung gian từ mã lệnh chương trình sử dụng cú pháp:

```
Debug. Print <danh_sách_các_biểu_thức>
```

- ✦ Thay đổi giá trị của một biến trong khi chạy chương trình từ cửa sổ trung gian. Chẳng hạn như trong chương trình đang thực thi có biến a, người lập trình có thể thay đổi giá trị của biến a thành 5 bằng cách gõ a=5 trong cửa sổ trung gian và nhấn phím ENTER.

### Cửa sổ theo dõi (Watch Window).

Để hiển thị cửa sổ theo dõi, trong VBA IDE chọn trình đơn **View** ⇒ **Watch Window**.



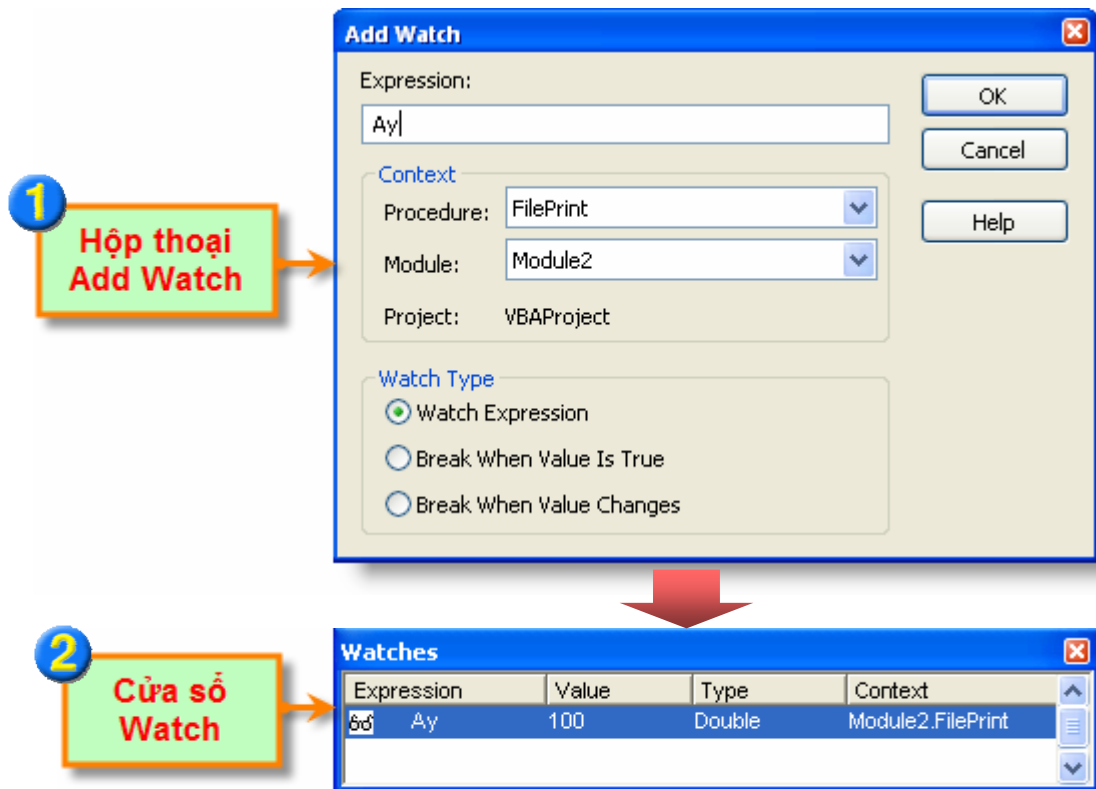
Hình III-29: Cửa sổ theo dõi.

Cửa sổ này thường được sử dụng để theo dõi sự biến đổi của các biến hoặc các biểu thức trong quá trình mã lệnh được thực thi. Ngoài ra, trong cửa sổ theo dõi, người lập trình có thể thay đổi giá trị cho biến trong lúc đang thực thi chương trình. Cần lưu ý là giá trị của biến/biểu thức cần theo dõi chỉ được hiển thị khi trình biên dịch đang thực thi một dòng lệnh nằm trong phạm vi hiệu lực của biến/biểu thức đó. Ví dụ như biến a trong chương trình con VD1 chỉ hiển thị giá trị trong cửa sổ theo dõi khi trình biên dịch đang thực thi một dòng lệnh nằm trong chương trình con VD1 đó.

Để thêm một biểu thức vào trong danh sách các biểu thức đang được theo dõi của cửa sổ Watch, thực hiện theo các bước sau:

1. Trong VBA IDE, chọn trình đơn **Debug** ⇒ **Add Watch** để hiển thị hộp thoại **Add Watch**.
2. Nhập biểu thức cần theo dõi trong mục **Expression**.
3. Chọn tên mô-đun và tên của chương trình con, nơi có chứa biến/biểu thức cần theo dõi trong mục **Module** và **Procedure**.
4. Nhấn ENTER hoặc chọn **OK** để thêm vào cửa sổ theo dõi.

**GỢI Ý** Để không phải thực hiện các bước 2 và 3, trước khi hiển thị cửa sổ theo dõi, cần đánh dấu chọn biến/biểu thức sẽ được theo dõi. Khi đó, các mục Expression, Module và Procedure sẽ được tự động điền các giá trị tương ứng.



Hình III-30: Thêm biểu thức vào cửa sổ theo dõi.

### 14.3. Bẫy lỗi trong VBAIDE

Như đã đề cập ở trên, khi gặp phải những lỗi phát sinh lúc thực thi chương trình sẽ gây ra những kết quả không thể tiên đoán được hoặc chương trình sẽ dừng lại và sẽ hiển thị thông báo lỗi rất phức tạp. Nếu đứng về phía người sử dụng chương trình thì những hộp thoại như vậy thường gây ra sự lúng túng khi sử dụng chương trình. Để tránh những hiện tượng như vậy, người lập trình cần phải thực hiện các kỹ thuật bẫy lỗi trong khi viết chương trình.

Bẫy lỗi thực chất là viết các đoạn mã lệnh chặn các thông báo lỗi mặc định của hệ thống và hướng dẫn chương trình cách thức xử lý lỗi đã chặn được. Các đoạn chương trình xử lý lỗi còn được gọi là bộ xử lý lỗi (error-handler). VBA có cung cấp các câu lệnh nhằm giúp người lập trình thực hiện bẫy lỗi trong chương trình của mình.

#### 14.3.1. Câu lệnh On Error

Câu lệnh `On Error` sẽ thực bật chế độ bẫy lỗi trong chương trình và xác định nơi sẽ thực hiện xử lý các lỗi khi lỗi xảy ra. Để tắt chế độ bẫy lỗi, người lập trình cũng dùng chính câu lệnh này. Các dạng cú pháp của câu lệnh này như sau:

Cú pháp	Mô tả
<code>On Error GoTo &lt;Label&gt;</code>	Bật chế độ bẫy lỗi. Khi có lỗi xảy ra, chương trình sẽ được tự động nhảy đến dòng lệnh có nhãn <Label> để tiếp tục thực thi mã lệnh. Đây chính là nơi chứa bộ xử lý lỗi của chương trình. Cần lưu ý là phần mã lệnh có nhãn <Label> phải nằm trong cùng một chương trình với câu lệnh <code>On Error</code> . Khi dùng bẫy lỗi kiểu này, trước nhãn <Label> thường có lệnh <code>Exit Sub</code> hoặc <code>Exit Function</code> (tùy thuộc chương trình con được bẫy lỗi) nhằm tránh thực thi bộ xử lý lỗi trong trường hợp lỗi không xảy ra.

On Error Resume Next	Bật chế độ bẫy lỗi. Khi có lỗi xảy ra, chương trình sẽ tự động nhảy đến dòng lệnh ngay sau dòng lệnh gây lỗi để tiếp tục thực thi mã lệnh. Câu lệnh này thường được sử dụng khi có câu lệnh truy xuất đến một đối tượng nào đó. Để nắm rõ lỗi đã phát sinh, câu lệnh này thường được sử dụng kết hợp với đối tượng Error (xem thêm mục “Đối tượng Err” trang 96)
On Error GoTo 0	Tắt chế độ bẫy lỗi. Khi thực hiện dòng lệnh này, các lỗi đã phát sinh trước đó sẽ được xoá và đồng thời kể từ sau dòng lệnh này, các lỗi sẽ không được chặn lại và xử lý nữa, và như vậy chương trình có thể ngưng hoạt động nếu có lỗi thực thi xảy ra.

Khi sử dụng câu lệnh `On Error GoTo <Label>`, ngay trước nhãn `<Label>` thường có lệnh `Exit Sub` hoặc `Exit Function` (tùy thuộc chương trình con được bẫy lỗi) nhằm tránh thực thi bộ xử lý lỗi trong trường hợp lỗi không xảy ra. Vì vậy, khuôn mẫu của các chương trình có bộ xử lý lỗi có thể được tham khảo thao đoạn mã lệnh sau:

```
Sub InitializeMatrix(Var1, Var2, Var3, Var4)
    On Error GoTo Bộ_xử_ly_lỗi
    . . .
    Exit Sub
Bộ_xử_ly_lỗi:
    . . .
    Resume Next
End Sub
```

Đoạn chương trình sau đây sẽ thực hiện truy xuất đến một tệp, sau đó đóng tệp đó lại. Nếu trong quá trình thao tác có lỗi xảy ra, chương trình sẽ được tự động nhảy đến dòng lệnh phía sau nhãn `lbErr` để hiển thị thông báo về lỗi đã xảy ra cho người sử dụng.

```
Sub SolveErrorExample()
    On Error GoTo lbErr
    Open "C:\fileABC.txt" For Input As 1
    Close 1
    Exit Sub
lbErr:
    MsgBox "Loi xay ra: " & Err.Description, vbCritical, "Thong bao loi"
End Sub
```

Khi thực thi chương trình, trong trường hợp tệp `C:\fileABC.txt` không tồn tại, người dùng sẽ nhận được thông báo lỗi như sau:



**Hình III-31: Thông báo lỗi do người dùng tự tạo**

### 14.3.2. Đối tượng Err

Đối tượng `Err` chứa tất cả các thông tin về lỗi thực thi của chương trình. Đối tượng này thường được sử dụng cùng với câu lệnh `On Error Resume Next`. Nhờ có đối tượng `Err` mà người lập trình có thể biết rõ được các thông tin về lỗi xảy ra để có hướng xử lý thích hợp.



Đối tượng `Err` có nhiều phương thức và thuộc tính khác nhau phục vụ cho việc xử lý lỗi. Trong đó, những thuộc tính và phương thức được sử dụng nhiều nhất bao gồm: `Description`, `Number` và `Clear`.

### Thuộc tính `Number`

Thuộc tính `Number` trả về số hiệu của lỗi thực thi. Đây cũng là thuộc tính mặc định của đối tượng `Err`, nghĩa là hai biểu thức `Err.Number` và `Err` là tương đương nhau, đều trả về số hiệu của lỗi thực thi.

Trong trường hợp không có lỗi xảy ra, thuộc tính này trả về giá trị 0.

### Thuộc tính `Description`

Thuộc tính `Description` trả về chuỗi ký tự mô tả thông tin ngắn gọn về lỗi thực thi đã xảy ra. Thông thường, khi lỗi xảy ra, nên ít nhất là hiển thị thông báo lỗi cho người dùng bằng cách sử dụng hàm `MsgBox` kết hợp với thuộc tính `Description`.

Trong trường hợp không có lỗi xảy ra, thuộc tính này trả về chuỗi ký tự rỗng "".

### Phương thức `Clear`

Phương thức `Clear` sẽ xóa tất cả các thuộc tính của đối tượng `Err`, có nghĩa là sau khi thực thi phương thức `Clear`, đối tượng `Err` sẽ được trở về trạng thái như khi không có lỗi xảy ra. Phương thức này thường được gọi sau khi đã tiến hành xử lý xong các lỗi thực thi.

Ví dụ sau sẽ minh họa cách thức sử dụng đối tượng `Err`. Trong ví dụ này có thực hiện phép chia cho 0, vì vậy chương trình sẽ làm phát sinh lỗi thực thi. Nhờ có câu lệnh `On Error GoTo out` nên khi có lỗi, chương trình sẽ tự động nhảy đến câu lệnh sau nhãn `out`. Vì vậy các câu lệnh sau câu lệnh làm phát sinh lỗi như `MsgBox x` và `Exit Sub` sẽ không bao giờ được thực hiện. Đoạn mã lệnh sau nhãn `out` thực hiện nhiệm vụ thông báo cho người dùng số hiệu lỗi và mô tả về lỗi đó

```
Sub test()
    On Error GoTo out

    Dim x, y
    x = 1 / y      ' Dòng lệnh này làm phát sinh lỗi chia cho 0
    MsgBox x
    Exit Sub
out:
    ' Hiển thị thông báo lỗi cho người dùng
    MsgBox "Ma loi: " & Err.Number
    MsgBox Err.Description
End Sub
```

### 14.3.3. Hàm `Error`

Hàm `Error` trả về chuỗi ký tự chứa mô tả về lỗi tương ứng của một số hiệu lỗi. Cú pháp của hàm như sau:

**`Error [ (errornumber) ]`**

Tham số `errornumber` là tham số tùy chọn, là số nguyên chứa số hiệu của một lỗi nào đó. Nếu `errornumber` là một lỗi hợp lệ nhưng chưa được định nghĩa, hàm `Error` sẽ trả về chuỗi "Application-defined or object-defined error.". Nếu `errornumber` là một số không hợp lệ thì sẽ làm phát sinh lỗi. Nếu tham số `errornumber` bị bỏ qua, hàm `Error` sẽ trả về mô tả của lỗi thực thi gần nhất.

Ví dụ sau sẽ hiển thị mô tả lỗi tương ứng của các số hiệu lỗi trong cửa sổ trung gian.

```
Sub VD_Error()  
  Dim ErrNumber  
  For ErrNumber = 61 To 64      ' Lặp qua các giá trị 61 - 64.  
    Debug.Print Error(ErrNumber)  ' In mô tả lỗi trong cửa sổ trung  
    gian.  
  Next ErrNumber  
End Sub
```

# CHƯƠNG IV: LẬP TRÌNH TRÊN MICROSOFT EXCEL

## 1. Tổng quan về Microsoft Excel

### 1.1. Khả năng của Excel

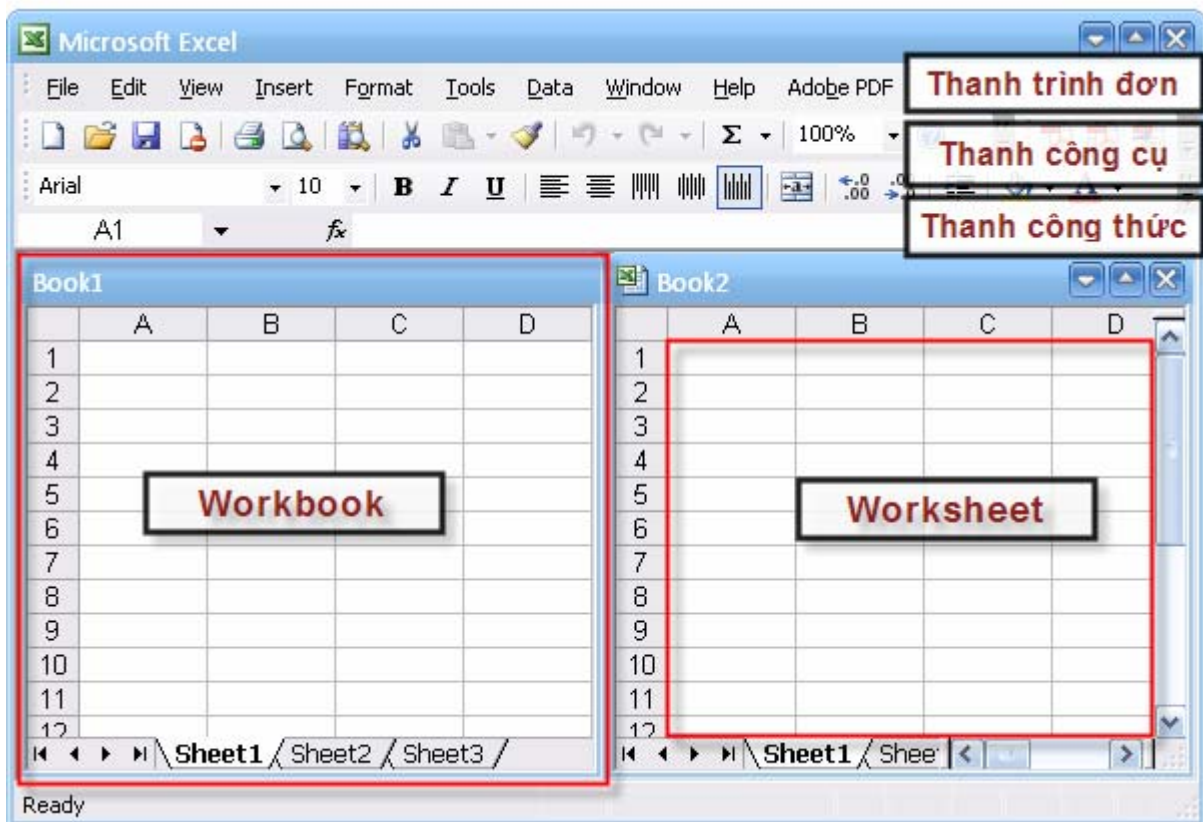
Microsoft Excel là một phần mềm chuyên xử lý bảng tính của hãng phần mềm nổi tiếng Microsoft. Excel thực sự là một công cụ rất mạnh mẽ phục vụ công tác tính toán, lập bảng biểu... Với các bài toán từ đơn giản đến phức tạp, ta đều có thể sử dụng Excel để giải quyết một cách dễ dàng với rất nhiều tính năng sẵn có:

- ◆ Khả năng tổ chức dữ liệu mạnh mẽ với hệ thống các ô, vùng dữ liệu, các bảng tính...;
- ◆ Khả năng xử lý dữ liệu như truy vấn, lọc, tính toán... với hệ thống rất phong phú các hàm cơ bản cũng như các hàm chức năng chuyên biệt;
- ◆ Khả năng lập báo cáo với cách tổ chức bảng biểu và hệ thống biểu đồ tương đối hoàn chỉnh;
- ◆ Khả năng in ấn với nhiều lựa chọn khác nhau.

Với cách tổ chức giống như bảng tính thông thường, Excel là một phần mềm bảng tính trực quan và rất dễ sử dụng. Chính bởi điều này khiến cho Excel là một trong những phần mềm được sử dụng phổ biến nhất.

### 1.2. Giao diện của Excel

Giao diện là nơi mà người dùng tương tác với chương trình và một giao diện hợp lý là giao diện quen thuộc với người dùng. Do chuyên về bảng tính, nên giao diện của Excel (như hình dưới) được thiết kế dựa trên sự mô phỏng của cấu trúc bảng tính thông thường.



Hình IV-1: Giao diện chính của Excel.

Các thành phần chính trong giao diện của Excel bao gồm:

1. **Thanh trình đơn** là nơi chứa các lệnh dùng để gọi đến các chức năng của chương trình. Hệ thống thanh trình đơn được truy cập bằng chuột, và trong một số lệnh phổ biến còn có thể sử dụng tổ hợp phím (ví dụ để lưu bảng tính có thể bấm phím Ctrl+S).
2. **Thanh công cụ** có rất nhiều thanh công cụ khác nhau, mỗi thanh công cụ chứa các nút lệnh trực quan hoặc các lựa chọn dùng để thực hiện một nhóm chức năng nào đó trong chương trình. Hay nói cách khác, một lệnh có thể được gọi từ thanh công cụ hoặc từ thanh trình đơn.
3. **Thanh công thức** bao gồm ô chứa địa chỉ của ô hiện hành và ô chứa nội dung của ô hiện hành. Tại đây ta có thể xem được công thức trong một ô nào đó trong khi ô đó vẫn chứa kết quả của công thức đó.
4. **Workbook** là một tệp tài liệu của Excel. Mỗi Workbook có thể chứa nhiều bảng tính (Worksheet) và các dữ liệu mở rộng khác. Tại mỗi thời điểm chỉ có một worksheet hiện hành và ta chỉ có thể làm việc với worksheet này.
5. **Worksheet** là loại tài liệu chính trong tệp tài liệu của Excel, mỗi worksheet chứa các ô tính (cell) được tổ chức thành các hàng và cột.

### 1.3. Khả năng mở rộng của Excel

Với hàng trăm hàm và rất nhiều lệnh có sẵn trong Excel khiến cho nó là một chương trình xử lý bảng tính rất mạnh, có thể giải quyết hầu hết các bài toán từ đơn giản đến phức tạp. Tuy vậy, việc lập trình mở rộng trên Excel vẫn luôn được đề cập đến, không những chỉ với mục đích là lập trình tạo thêm những tính năng mới cho Excel mà còn để kết hợp các tính năng sẵn có của chương trình Excel để giải quyết những vấn đề mang tính chuyên biệt hoá cao.

Bộ chương trình **Dự Toán** là một ví dụ cụ thể cho việc lập trình mở rộng trên Excel. Các bài toán chuyên biệt về tính toán dự toán công trình đã được giải quyết một cách dễ dàng dựa trên sự kết hợp giữa các hàm có sẵn trong Excel và một số tính năng mới về cơ sở dữ liệu.

Việc lập trình mở rộng Excel có thể được thực hiện theo nhiều cách khác nhau, nhưng đơn giản và hiệu quả nhất có thể kể đến những cách sau:

- ✦ Lập trình mở rộng thông qua môi trường lập trình VBAIDE được tích hợp sẵn trong Excel. Theo cách này, người dùng sẽ sử dụng ngôn ngữ lập trình VB để lập trình mở rộng Excel. Các ứng dụng được tạo ra theo cách này gắn liền với tệp tài liệu của Excel (Workbook).
- ✦ Lập trình mở rộng thông qua bộ công cụ lập trình Visual Studio Tools for Office (VSTO) trong bộ công cụ phát triển phần mềm Microsoft Visual Studio. Theo cách này, người sử dụng có thể lập trình tạo ra các ứng dụng chuyên nghiệp dạng Add-in (ứng dụng bổ sung trong Excel) bằng các ngôn ngữ được hỗ trợ trong Microsoft Visual Studio. Ứng dụng mở rộng dạng này được lưu trữ tách biệt với tệp tài liệu của Excel nên rất dễ dàng phân phối.

Với những ưu điểm vốn có của VBA và cùng với khả năng sẵn có của Excel, hầu hết các bài toán trong lĩnh vực thiết kế công trình giao thông đều có thể giải quyết được thông qua việc lập trình mở rộng Excel. Vì vậy, trong toàn bộ tài liệu này, việc lập trình mở rộng Excel sẽ được đề cập đến theo cách dựa trên môi trường lập trình VBAIDE.

Để khởi động VBAIDE, từ cửa sổ chính của Excel, chọn trình đơn **Tools⇒Macro⇒Visual Basic Editor**, hoặc có thể sử dụng tổ hợp phím **ALT+F11**.

## 2. Macro

Cách tốt nhất để làm quen với việc lập trình trên Excel chính là sử dụng Macro và tìm hiểu cách thức hoạt động của nó.

## 2.1. Macro là gì?

Khi làm việc trong Excel, đôi lúc gặp phải 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, ví dụ như thường xuyên phải định dạng dữ liệu thành một kiểu bảng giống nhau. Điều này rất dễ dẫn đến sự nhầm lẫn trong công việc. Do đó, khi thiết kế Excel, Microsoft đã đưa ra khái niệm Macro để có thể gói gọn tất cả các 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 (cách gọi: chọn trình đơn **Tools** ⇒ **Macro** ⇒ **Macros** hoặc bấm **Alt+F8**).

Trong các khai báo chương trình con trong ví dụ sau, ta sẽ thấy được cách định nghĩa một Macro:

```
Sub Macro()                \←Macro
Public Sub Macro ()       \←Macro
Private Sub Macro()      \←Thủ tục với từ khoá Private, không phải Macro
Sub Macro(Input as Double) \←Thủ tục có tham số, không phải Macro
Public Function Macro() as Double \←Hàm, không phải Macro
```

## 2.2. Tạo Macro

### 2.2.1. Tạo Macro theo kịch bản

Đây là cách tạo Macro dễ dàng nhất, theo cách này, người sử dụng sẽ chuẩn bị trước tất cả các thao tác sẽ thực hiện (xây dựng một kịch 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.



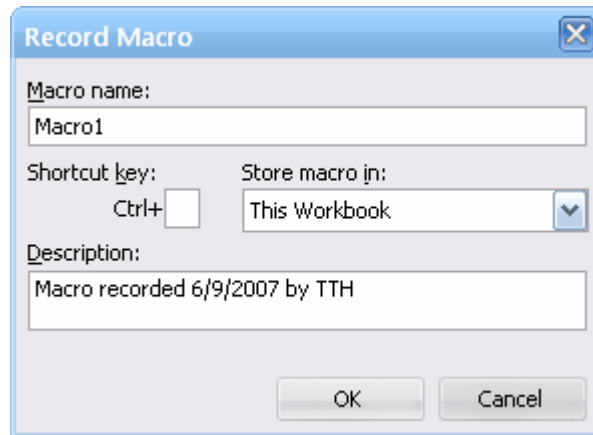
**CHÚ Ý** Nếu trong quá trình thu Macro, người sử dụng thực hiện không đúng theo kịch bản dự định (bị lỗi) và có thêm những thao tác để sửa lại các lỗi đó, thì toàn bộ những thao tác phát sinh này cũng sẽ được ghi nhận như là một phần của Macro.

Ví dụ sau sẽ tiến hành thu Macro có nhiệm vụ định dạng một bảng dữ liệu với định dạng như sau:

Tiêu đề	Tiêu đề	Tiêu đề	Tiêu đề
Nội dung	Nội dung	Nội dung	Nội dung
Nội dung	Nội dung	Nội dung	Nội dung

**Tạo Macro theo kịch bản dùng để định dạng bảng dữ liệu:**

1. Chọn vùng dữ liệu cần định dạng, ví dụ vùng A1:D5.
2. Trong trình đơn **Tools**, chọn **Macro**⇒**Record New Macro...** Để hiển thị hộp thoại **Record Macro**.



**Hình IV-2: Hộp thoại Record Macro.**




3. Trong mục **Macro name**, nhập tên của Macro, ví dụ là Macro1.
4. Nếu muốn thực thi Macro bằng cách nhấn phím tắt, nhập một chữ cái thông thường vào ô **Shortcut Key**. Sau đó, để thực thi Macro, ta chỉ cần nhấn tổ hợp phím CTRL+Chữ cái (với chữ viết thường) hoặc CTRL+SHIFT+Chữ cái (với chữ viết hoa). Chữ cái đặt làm phím tắt không được phép là số hay các ký tự đặc biệt như @ hoặc #. Nếu phím tắt này trùng với các phím tắt đã có thì những phím tắt đã có sẽ bị vô hiệu hoá.
5. Trong mục **Store Macro In**, chọn nơi sẽ lưu trữ Macro. Nếu muốn Macro có thể sử dụng được ngay cho mọi bảng tính mỗi khi sử dụng Excel, thì chọn mục **Personal Macro Workbook**. Trong ví dụ này, chọn This Workbook.




**CHÚ Ý** Nếu người dùng tạo một Macro khá hữu dụng và muốn dùng lại nhiều lần thì nên chọn lưu Macro trong Personal Macro Workbook. Tập bảng tính này là một tập bảng tính ẩn có tên là Personal.xls, được lưu trong thư mục Xlstart. Mỗi khi khởi động Excel, tập bảng tính này sẽ được tự động tải lên nhưng ở chế độ ẩn. Mặc định, tập Personal.xls không tồn tại cho đến khi người dùng tạo Macro và Macro đó được lưu vào Personal Macro Workbook (chọn trong Store Macro In của hộp thoại Record Macro).

6. Nhập các thông tin vào mục **Description** nếu cần mô tả thêm về Macro này.
7. Chọn **OK**.



**CHÚ Ý** Trong quá trình tạo Macro kịch bản, nếu muốn lưu địa chỉ ô tương đối so với ô hiện hành, ta làm như sau: trên thanh công cụ Stop Recording , chọn vào biểu tượng tương đối Reference . Kể từ thời điểm ấy, địa chỉ ô sẽ được lưu tương đối so với ô hiện hành cho đến khi thoát khỏi Excel hoặc chọn một lần nữa vào biểu tượng Relative Reference .

8. Thực hiện các thao tác mà sau này sẽ được lặp lại khi Macro kịch bản thực thi.
  - a. Định dạng các đường kẻ cho bảng dữ liệu: Chọn trình đơn **Format**⇒**Cells...** ⇒ Chọn thẻ **Border** để định dạng các đường kẻ cho bảng dữ liệu.

- b. Định dạng dòng tiêu đề của bảng dữ liệu: Chọn dòng đầu tiên của bảng dữ liệu ⇒ Chọn trình đơn **Format** ⇒ **Cells...** ⇒ Chọn thẻ **Font** ⇒ chọn **Font Style** là **Bold** ⇒ Chọn thẻ **Partern** ⇒ Chọn màu xám.
9. Trên thanh công cụ **Stop Recording**, nhấn chuột vào biểu tượng Stop Recording  để hoàn thành việc tạo Macro theo kịch bản.

Sau khi kết thúc quá trình tạo Macro theo kịch bản, Excel sẽ tự động phát sinh một đoạn mã lệnh như sau:

```
Sub Macro1()                                '←Tên Macro
'
' Macro1 Macro
' Macro recorded 6/10/2007 by TTH
'
' Keyboard Shortcut: Ctrl+Shift+L           ←Phím tắt của Macro
'
    Selection.Borders(xlDiagonalDown).LineStyle = xlNone
    Selection.Borders(xlDiagonalUp).LineStyle = xlNone
    With Selection.Borders(xlEdgeLeft)
        .LineStyle = xlContinuous
        .Weight = xlMedium
        .ColorIndex = xlAutomatic
    End With
    With Selection.Borders(xlEdgeTop)
        .LineStyle = xlContinuous
        .Weight = xlMedium
        .ColorIndex = xlAutomatic
    End With
    With Selection.Borders(xlEdgeBottom)
        .LineStyle = xlContinuous
        .Weight = xlMedium
        .ColorIndex = xlAutomatic
    End With
    With Selection.Borders(xlEdgeRight)
        .LineStyle = xlContinuous
        .Weight = xlMedium
        .ColorIndex = xlAutomatic
    End With
    With Selection.Borders(xlInsideVertical)
        .LineStyle = xlContinuous
        .Weight = xlThin
        .ColorIndex = xlAutomatic
    End With
    With Selection.Borders(xlInsideHorizontal)
        .LineStyle = xlContinuous
        .Weight = xlThin
        .ColorIndex = xlAutomatic
    End With
    ActiveWindow.SmallScroll Down:=-6
    Range("A1:D1").Select                    '←Lựa chọn hàng tiêu đề
    With Selection.Font                       '←Phông chữ cho hàng tiêu đề
        .Name = "Arial"
        .FontStyle = "Bold"
        .Size = 10
        .Strikethrough = False
        .Superscript = False
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
```

```

        .Underline = xlUnderlineStyleNone
        .ColorIndex = xlAutomatic
    End With
    With Selection.Interior                '← Tô màu cho hàng tiêu đề
        .ColorIndex = 48
        .Pattern = xlSolid
        .PatternColorIndex = xlAutomatic
    End With
End Sub                                    '← Kết thúc Macro
    
```

Đoạn mã trên đã được thêm vào một vài dòng ghi chú để làm rõ hơn từng phần của Macro, mỗi ghi chú bắt đầu bằng dấu '←.

## 2.2.2. Tạo Macro sử dụng 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 về thao 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 để tạo 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 VBAIDE.

Ví dụ sau minh họa cách thức tạo một Macro sử dụng VBA. Mục đích của Macro là định dạng lại phông chữ cho vùng ô đang được lựa chọn trong bảng tính: thay đổi tên phông chữ thành “Time News Roman”, kiểu chữ thành “Italic”, kích cỡ chữ “11”.

### Tạo Macro sử dụng VBAIDE

1. Trong màn hình chính của Excel, chọn trình đơn **Tools⇒Macro⇒Visual Basic Editor**.
2. Trong màn hình của VBAIDE vừa được hiển thị, chọn trình đơn **Insert⇒Module**.
3. Nhập đoạn mã lệnh sau:

```

Sub Dinh_dang()
    With Selection.Font
        .Name = "Times New Roman"
        .FontStyle = "Italic"
        .Size = 11
    End With
End Sub
    
```

4. Sau khi nhập xong đoạn mã lệnh, chọn trình đơn **File⇒Close and Return to Microsoft Excel** để trở về màn hình chính của Excel.



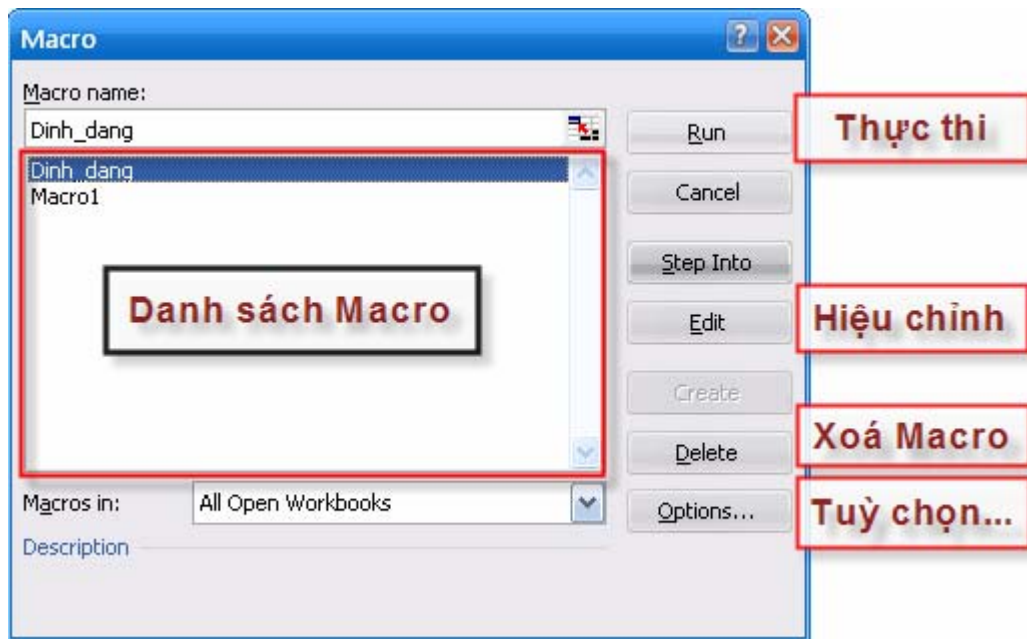
**CHÚ Ý** Mỗi Macro đều có một tên riêng và tên này là duy nhất trong một tài liệu Excel (Workbook).

## 2.3. Quản lý Macro

Nhằm tạo điều kiện thuận lợi cho người sử dụng trong khi làm việc với Macro, Excel đã tích hợp sẵn một trình quản lý Macro.

Để hiển thị trình quản lý Macro, chọn trình đơn **Tools⇒Macro⇒Macros...** hoặc nhấn tổ hợp phím **ALT+F8**.

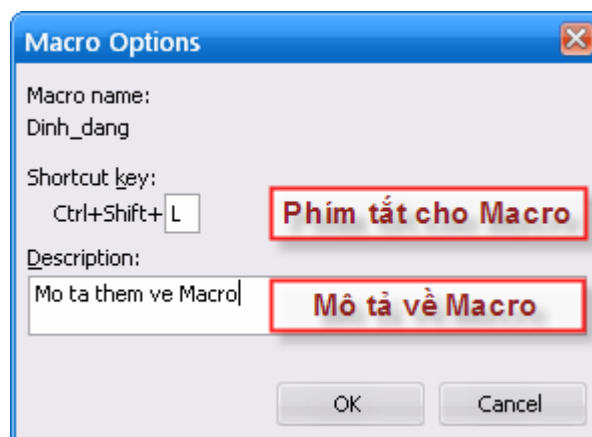




Hình IV-3: Trình quản lý Macro

Trong cửa sổ Macro, các Macro được tạo theo kịch bản hoặc bằng VBAIDE có trong phiên làm việc hiện tại của Excel sẽ được hiển thị trong một danh sách. Tất cả các thao tác quản lý Macro sẽ được thực hiện dễ dàng thông qua trình quản lý này. Để bắt đầu một thao tác nào đó, trước hết cần phải chọn Macro tương ứng có trong danh sách:

- ◆ Để thực thi Macro (chạy Macro): kích chuột vào nút **Run**.
- ◆ Để hiệu chỉnh Macro: kích chuột vào nút **Edit**, cửa sổ lệnh trong VBAIDE chứa các mã lệnh của Macro được chọn sẽ được hiển thị để người sử dụng có thể thay đổi mã lệnh trong Macro đó.
- ◆ Để xoá Macro: kích chuột vào nút **Delete**, Macro được chọn sẽ được xoá cả trong danh sách Macro và mã lệnh của Macro đó.
- ◆ Kích chuột vào nút **Options...** sẽ hiển thị hộp thoại lựa chọn, cho phép người sử dụng thiết lập lại phím tắt hoặc thay đổi mô tả cho Macro được chọn.



Hình IV-4: Hộp thoại Macro Options.

## 2.4. Sử dụng Macro

Việc sử dụng các Macro đã được tạo, thực chất là thực thi đoạn mã lệnh tạo nên Macro đó. Có nhiều cách khác nhau để chạy một Macro:

- ◆ Thực thi bằng cách bấm phím tắt đã gán cho Macro;

- ◆ Thực thi Macro theo cách thông qua trình quản lý Macro;
- ◆ Thực thi Macro trực tiếp từ VBAIDE;
- ◆ Thực thi bằng cách nhấn chuột vào một nút lệnh hay một điều khiển đồ hoạ mà đã được gán trở tới Macro cần thực hiện;
- ◆ Thực thi bằng cách nhấn chuột vào một đối tượng đồ hoạ mà đã được gán trở tới Macro;
- ◆ Thực thi thông qua nút lệnh trên thanh công cụ;
- ◆ Thực thi thông qua mục trên thanh trình đơn.

Chi tiết về các cách thực thi Macro xin tìm hiểu thêm trong tài liệu “Microsoft Office Excel Help” được cài đặt sẵn cùng Excel. Ở đây chỉ trình bày cách thực thi Macro theo một số cách thông thường.

### 2.4.1. Thực thi Macro bằng phím tắt

Trong quá trình tạo Macro theo kịch bản, 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. Trong ví dụ ở phần “*Tạo Macro theo kịch bản*” trang 101, Macro đã được gán một tổ hợp phím tắt là CTRL+SHIFT+L, do vậy, để thực thi Macro này, người sử dụng chỉ cần chọn vùng dữ liệu để định dạng bảng, sau đó nhấn tổ hợp phím CTRL+SHIFT+L.

Đối với Macro được tạo bằng cách sử dụng VBAIDE, người dùng chỉ có thể tạo phím tắt cho Macro thông qua trình quản lý Macro. Chi tiết tham khảo phần “*Quản lý Macro*” trang 104.


### 2.4.2. Thực thi Macro thông qua trình quản lý Macro

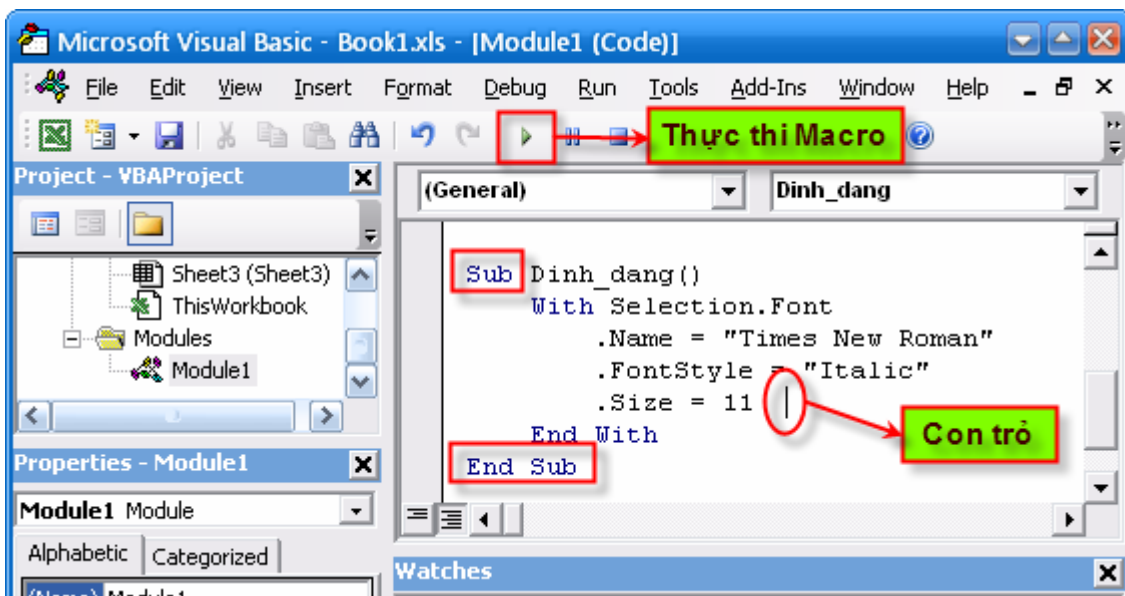
Chi tiết tham khảo phần “*Quản lý Macro*” trang 104.

### 2.4.3. Thực thi Macro trực tiếp từ VBAIDE

Cách thực thi Macro trực tiếp từ VBAIDE rất thích hợp khi người sử dụng muốn thử nghiệm ngay Macro trong quá trình xây dựng nó.

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

1. Trong cửa sổ mã lệnh của VBAIDE, đặt con trỏ vào giữa khối **Sub ... End Sub**.
2. Nhấn phím **F5** hoặc chọn biểu tượng  trên thanh công cụ.



Hình IV-5: Thực thi Macro trực tiếp từ VBAIDE

Trong trường hợp người sử dụng không đặt con trỏ giữa, một danh sách các Macro sẽ được hiện ra để người dùng lựa chọn Macro cần thực thi.

### 2.5. Hiệu chỉnh Macro

Khi Macro được tạo ra chưa đáp ứng đủ nhu cầu thì người sử dụng có thể thay đổi, bổ sung mã lệnh cho Macro đó. Quá trình hiệu chỉnh Macro được thực hiện thông qua VBAIDE. Để hiệu chỉnh Macro, ta có thể dùng trình quản lý Macro (xem mục “*Quản lý Macro*” trang 104) hoặc truy cập trực tiếp trong VBAIDE. Về bản chất, việc hiệu chỉnh (sửa đổi) Macro tương đương như việc lập trình để xây dựng nên Macro đó.

### 2.6. Vấn đề an toàn khi sử dụng Macro

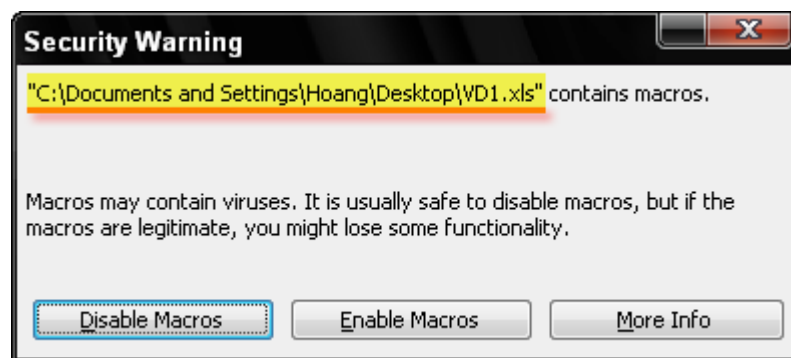
Do Macro là những đoạn mã lệnh có thể tự động thực thi và những đoạn mã lệnh này có thể gây nguy hiểm cho máy tính của người dùng (dạng Macro Virus). Chính vì vậy, Excel sử dụng cơ chế bảo vệ để chống lại nguy cơ lây nhiễm virus thông qua Macro. Cơ chế này có thể được điều chỉnh thông qua các mức an ninh khác nhau:

- ◆ Very High
- ◆ High
- ◆ Medium
- ◆ Low



**GỢI Ý** Mức an ninh của Excel có thể được thiết lập bằng cách chọn trình đơn **Tools**⇒**Macro**⇒**Security...**

Thông thường, khi sử dụng Excel với các tệp bảng tính có chứa Macro, nên đặt mức an ninh ở **Medium**. Ở mức này, Excel sẽ yêu cầu người dùng xác thực xem các đoạn mã lệnh trong tệp bảng tính có phải từ nguồn tin cậy hay không.



**Hình IV-6: Hộp thoại cảnh báo an ninh của Excel**

Nếu người dùng chọn Enable Macros, các Macro chứa trong workbook đó sẽ được phép thực thi.

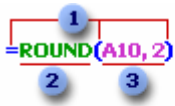
Nếu người dùng chọn Disable Macros, các Macro chứa trong workbook đó vẫn tồn tại trong workbook nhưng không thể thực thi được.

## 3. Xây dựng hàm mới trong Excel

### 3.1. Khái niệm về hàm trong Excel

Hàm là những công thức đã được định nghĩa sẵn trong Excel để thực hiện tính toán dựa trên các số liệu đầu vào, gọi là tham số, theo một trình tự đã được lập trình sẵn nhằm thực hiện các phép tính từ đơn giản đến phức tạp.

Để hiểu rõ hơn về cấu trúc của một hàm, ta tìm hiểu về hàm ROUND có sẵn trong Excel, là hàm dùng để làm tròn số:



`ROUND(number, num_digits)`

1. Cấu trúc. Một hàm bắt đầu bằng dấu bằng "=", tiếp sau là tên hàm, dấu ngoặc đơn "(" , danh sách các tham số cách nhau bằng dấu phẩy "," và cuối cùng là dấu ngoặc đơn ")" .
2. Tên hàm. Ấn phím SHIFT+F3 để hiển thị danh sách tất cả các hàm trong Excel.
3. Các tham số. Tham số có thể là số, chữ, giá trị logic như TRUE hoặc FALSE, mảng, giá trị lỗi như #NA, hoặc tham chiếu đến một ô khác. Tham số truyền vào phải có kiểu thích hợp với kiểu của từng tham số tương ứng của hàm. Tham số truyền vào có thể là một hằng số, công thức, hoặc là một hàm bất kỳ.
4. Chú thích hàm. Chú thích hàm dùng để thể hiện cấu trúc và danh sách các tham số của hàm, hiện lên khi ta nhập vào tên hàm. Chú thích hàm chỉ xuất hiện đối với những hàm được xây dựng sẵn trong Excel.

### 3.2. Tạo hàm mới bằng VBA

#### 3.2.1. Tại sao phải dùng hàm?

Trong quá trình tính toán với các bảng tính, người ta thấy rằng luôn tồn tại một nhu cầu: giá trị trong một ô nào đó sẽ được tính dựa vào hai yếu tố:

- ◆ Tính theo một hoặc nhiều công thức hoặc theo một trình tự logic nào đó.
- ◆ Việc tính toán cần phải dựa trên những thông số bên ngoài khác.

Nếu đối chiếu hai yếu tố trên với cấu trúc của một hàm (ở mục trước) ta có thể thấy rằng nhu cầu trên chỉ có thể được giải quyết một cách thỏa đáng với việc sử dụng hàm. Hơn nữa, khi sử dụng hàm, việc sử dụng lặp cho nhiều ô hoặc hiệu chỉnh nội dung tính toán sau này đều rất thuận tiện so với việc không dùng hàm (tính trực tiếp trong ô). Chính bởi ưu điểm này mà hàng loạt hàm đã được Excel xây dựng sẵn và phân loại theo nhóm để tạo thuận tiện cho người dùng.

Một câu hỏi đặt ra là với hơn 300 hàm có sẵn trong Excel cộng với các hàm có sẵn trong VBA, tại sao lại cần phải tạo ra hàm mới? Câu trả lời rất đơn giản: để đơn giản hoá công việc. Với một chút sáng tạo, người dùng có thể tạo thêm các hàm mới phục vụ cho những nhu cầu của mình.

Không phải lúc nào các hàm có sẵn cũng có thể giải quyết được công việc của người dùng, hoặc có thể giải quyết được nhưng phải thông qua rất nhiều hàm khác nhau hoặc thực hiện theo một cách rất phức tạp. Thay vào đó, người dùng có thể tạo ra một hàm mới đảm nhận nhiệm vụ này. Hàm mới này có thể có cách thức tính toán hoàn mới, hoặc cũng có thể chỉ là việc tập hợp lại các hàm sẵn có để tạo thành một hàm đơn giản hơn. Càng đơn giản, càng dễ hiểu, dễ nhớ và dễ sử dụng.

Lấy ví dụ như trong Excel, có cung cấp một bộ công cụ có tên là **Lookup**. Bộ công cụ này cho phép người sử dụng tiến hành tra bảng 2 chiều một cách dễ dàng. Nhưng việc tra bảng và nội suy không thể thực hiện được nhờ bộ công cụ này. Vì thế, xây dựng một hàm mới dùng để tra bảng và nội suy 2 chiều sẽ là một công cụ tốt phục vụ cho quá trình tính toán, nhất là đối với ngành công trình.

Excel cho phép xây dựng các hàm mới bằng VBA, và đặc biệt, việc sử dụng các hàm mới này không khác gì so với việc sử dụng các hàm có sẵn của Excel. Hàm mới luôn mang đặc tính:

- ✦ Trả về một giá trị nào đó, tương tự như hàm có sẵn trong Excel;
- ✦ Hàm mới có thể sử dụng như một chương trình con trong VBA, nghĩa là nó vừa có thể sử dụng trong bảng tính (trong các ô), đồng thời có thể sử dụng trong các chương trình viết bằng VBA.

### 3.2.2. Cấu trúc hàm

Thực chất, hàm là một chương trình con dạng Function. Khác với Macro, hàm là chương trình con có giá trị trả về và có thể có tham số.

Khi tạo hàm mới, người sử dụng cần phải tuân thủ theo dạng thức khai báo như sau:

```
[Public/Private] Function Tên_hàm([DSách_tham_số]) [as kiểu_dữ_liệu]
    [Câu_lệnh]
    [Tên_hàm = biểu_thức]
    [Exit Function]
    [Câu_lệnh]
    [Tên_hàm = biểu_thức]
End Function
```

Trong đó:

- ✦ **Public:** (tùy chọn) là từ khoá biểu thị phạm vi của hàm, hàm có thể được sử dụng ở bất kỳ đâu trong tất cả các dự án VBA hiện có. Khi có từ khoá Public, tên hàm sẽ được hiển thị trong danh sách hàm của Excel.
- ✦ **Private:** (tùy chọn) là từ khoá biểu thị phạm vi của hàm, hàm chỉ có thể được sử dụng bên trong mô-đun có chứa hàm đó. Khi có từ khoá Private, tên hàm sẽ **không** được hiển thị trong danh sách hàm của Excel, nhưng người sử dụng vẫn có thể dùng hàm này trong bảng tính một cách bình thường.



**CHÚ Ý** Nếu không khai báo phạm vi cho hàm (từ khoá Public/Private), thì mặc định, hàm sẽ có phạm vi là Public.

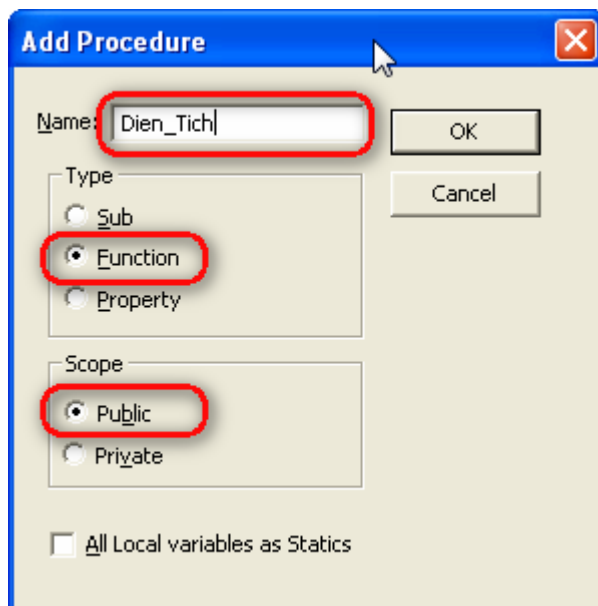
- ✦ **Function:** (bắt buộc) là từ khoá báo hiệu bắt đầu một hàm.
- ✦ **Tên\_hàm:** (bắt buộc) là tên của hàm, cách đặt tên hàm tương tự như cách đặt tên của biến. Tên\_hàm sẽ được sử dụng như là biến trong toàn bộ hàm, khi hàm kết thúc giá trị trả về của hàm chính là giá trị đã gán cho biến Tên\_hàm cuối cùng.
- ✦ **Danh\_sách\_tham\_số:** (tùy chọn) là danh sách các tham số đầu vào của hàm. Các tham số được phân cách với nhau bằng dấu phẩy.
- ✦ **Kiểu\_dữ\_liệu:** (tùy chọn) quy định kiểu giá trị trả về của hàm. Nếu không quy định kiểu dữ liệu, hàm sẽ có kiểu dữ liệu mặc định là Variant.
- ✦ **Exit Function:** (tùy chọn) là câu lệnh dùng để kết thúc hàm ngay lập tức (cho dù phía sau câu lệnh này vẫn còn các khối lệnh khác).
- ✦ **End Function:** (bắt buộc) là từ khoá báo hiệu kết thúc một hàm.

### 3.2.3. Tạo hàm mới

Để minh họa rõ hơn cách thức tạo hàm mới, lấy ví dụ tạo một hàm rất đơn giản: hàm tính diện tích hình chữ nhật. Hàm này có tên là `Diện_tích`, với hai tham số đầu vào là chiều rộng và chiều cao. Kiểu dữ liệu của các tham số là kiểu số thực và giá trị trả về của hàm cũng là kiểu số thực.

**Để tạo một hàm mới, thực hiện theo các bước sau:**

1. Khởi động VBAIDE. Trong trình đơn **Tools**, chọn mục **Macro**⇒**Visual Basic Editor**;
2. Trong trình đơn **Insert**, chọn mục **Module** để tạo một mô-đun mới, nơi sẽ chứa hàm do người dùng định nghĩa.
3. Trong trình đơn **Insert**, chọn mục **Procedure...** để hiển thị hộp thoại **Add Procedure**. Sau đó điền tên hàm vào mục **Name**, chọn kiểu chương trình con là **Function** và phạm vi là **Public**. Cuối cùng chọn **OK**;



**Hình IV-7: Hộp thoại Add Procedure.**

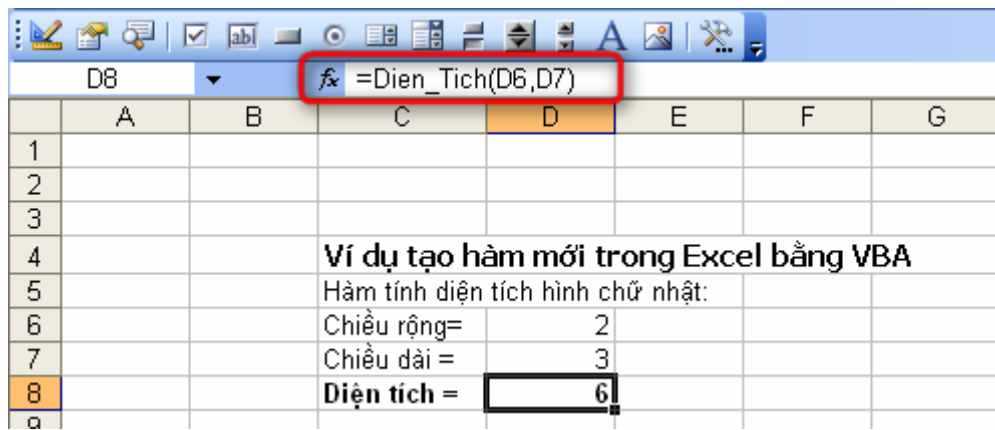
4. Chương trình sẽ tự động phát sinh đoạn mã lệnh như sau:

```
Public Function Dien_Tich()  
End Function
```

5. Thay đoạn mã lệnh trên bằng đoạn mã lệnh sau:

```
Public Function Dien_Tich(Rong As Double, Cao As Double) As Double  
    'Hàm tính diện tích hình chữ nhật  
    Dien_Tich = Rong * Cao  
End Function
```

6. Trong trình đơn **File**, chọn mục **Close and Return to Microsoft Excel** để quay trở về màn hình chính của Excel;
7. Lúc này, hàm mà ta vừa xây dựng, có tên là **Dien\_Tich**, đã có thể được sử dụng bình thường như các hàm khác của Excel.



Hình IV-8: Sử dụng hàm mới trong Excel.



**CHÚ Ý** Các bước tạo hàm mới cũng tương tự như các bước tạo Macro ở phần trước. Tuy nhiên, do hàm cần phải có giá trị trả về nên khi khai báo kiểu chương trình con cho hàm, người dùng cần phải chọn là **Function** (khác với khi tạo Macro, phải chọn là **Sub**).

Thông thường, với yêu cầu tính toán trên không nhất thiết phải tạo hàm mới, đây chỉ là một đoạn ví dụ rất đơn giản nhằm minh họa cách thức tạo hàm và cấu trúc của hàm. Để hiểu rõ hơn về hàm, ta cùng xem lại đoạn mã trên:

```
Public Function Dien_Tich(Rong As Double, Cao As Double) As Double
    'Hàm tính diện tích hình chữ nhật
    Dien_Tich = Rong * Cao
End Function
```

Ở dòng đầu tiên, được bắt đầu bằng từ khoá `Public`, do vậy tên hàm sẽ được hiển thị trong danh sách hàm trong Excel (được hiển thị khi nhập dấu bằng vào ô và bấm phím `Shift+F3`). Tiếp sau đó là từ khoá `Function` (chứ không phải là `Sub` như Macro) và tên hàm, `Dien_Tich`. Hàm có hai tham số, nằm giữa hai dấu ngoặc đơn, là `Rong` và `Cao`, và đều có kiểu số thực. Từ khoá `As Double` ở cuối xác định kiểu trả về của hàm `Dien_Tich` là kiểu số thực.

Ở dòng thứ 2, đơn giản chỉ là một dòng chú thích vì được bắt đầu bằng dấu phẩy trên (`'`).

Ở dòng thứ 3, giá trị của hàm được tính dựa trên hai tham số đầu vào là `Rong` và `Cao`.

Hàm được kết thúc bằng câu lệnh `End Function`.



**CHÚ Ý** Khi xây dựng hàm mới, cần phải chú ý sự khác biệt giữa hàm gọi từ các chương trình con trong VBA và hàm sử dụng trong bảng tính. Các hàm sử dụng trong bảng tính mang tính "bị động", tức là không thể thao tác trên các vùng dữ liệu hoặc thay đổi nội dung nào đó trong bảng tính.

Nếu người dùng cố tạo một hàm mà trong đó có thay đổi định dạng của một ô, như màu nền chẳng hạn, thì những hàm như vậy sẽ không thực hiện được, và hàm sẽ luôn trả về giá trị lỗi.

Như vậy, khi tạo hàm mới cần ghi nhớ: Hàm chỉ đơn giản là trả về một giá trị nào đó; Hàm không thể thực hiện thao tác làm thay đổi đối tượng.

### 3.3. Hàm trả về lỗi

Trong một số trường hợp, hàm có thể sẽ phải trả về một giá trị lỗi nào đó. Để làm rõ hơn điều này, lấy ví dụ hàm phân loại sinh viên.

```
Function PhanLoai(DiemTB) As String
    If (DiemTB >= 5) Then
        PhanLoai = "Đo"
```

```
Exit Function
End If

If (DiemTB < 5) Then
    PhanLoai = "Truot"
    Exit Function
End If
End Function
```

Hàm này lấy tham số đầu vào để phân loại là điểm trung bình của sinh viên thông qua biến DiemTB với thang điểm 10. Hàm sẽ trả về giá trị kiểu chuỗi: “Đỗ” nếu điểm trung bình lớn hơn hoặc bằng 5, và ngược lại là “Trượt”.

Tuy nhiên, với những trường hợp điểm trung bình, vì một lý do nào đó, được nhập vào giá trị nhỏ hơn 0 hoặc lớn hơn 10 thì hàm vẫn trả về thông báo “Trượt” hoặc “Đỗ”. Như vậy là không hợp lý. Trong những trường hợp đó, cần phải trả về thông báo cho người sử dụng biết là không thể áp dụng hàm với điểm trung bình như vậy. Như vậy, hàm sẽ được thay đổi lại như sau:

```
Function PhanLoai(DiemTB) As String
    If (DiemTB < 0) Or (DiemTB > 10) Then
        PhanLoai = "#N/A"
        Exit Function
    End If

    If (DiemTB >= 5) Then
        PhanLoai = "Do"
        Exit Function
    End If

    If (DiemTB < 5) Then
        PhanLoai = "Truot"
        Exit Function
    End If
End Function
```

Và như vậy, kể từ lúc này, mỗi khi vô tình nhập các giá trị điểm không thích hợp, hàm sẽ trả về một thông báo lỗi là “#N/A”. Mặc dù trông rất giống lỗi trong Excel, nhưng thực chất đây vẫn chỉ là một chuỗi thông thường.

Để trả về giá trị lỗi thực sự, VBA đã cung cấp thêm một hàm tên là CVErr, hàm này sẽ chuyển đổi một số thành một giá trị lỗi tương ứng. Với giá trị lỗi thực sự như thế, tất cả những hàm có tham chiếu ô chứa giá trị lỗi cũng sẽ trả về giá trị lỗi tương tự. Và như vậy, người dùng chỉ cần thay đổi câu lệnh PhanLoai = “#N/A” bằng câu lệnh PhanLoai = CVErr(xlErrNA). Cần lưu ý là kiểu trả về của hàm CVErr là kiểu Variant, do vậy cũng cần phải thay đổi kiểu giá trị trả về của hàm là Variant.

Hàm sẽ được hiệu chỉnh lại như sau:

```
Function PhanLoai(DiemTB) As Variant
    If (DiemTB < 0) Or (DiemTB > 10) Then
        PhanLoai = CVErr(xlErrNA)
        Exit Function
    End If
    If (DiemTB >= 5) Then
        PhanLoai = "Do"
        Exit Function
    End If
```



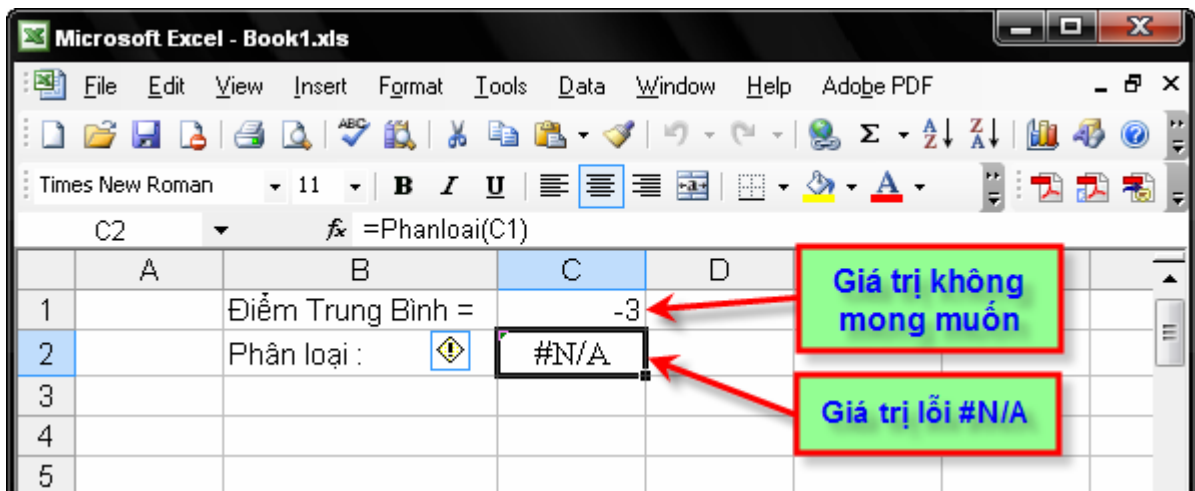
```

If (DiemTB < 5) Then
    PhanLoai = "Truot"
Exit Function
End If
End Function
    
```



**CHÚ Ý** Để sử dụng hàm trả về thông báo lỗi, nghĩa là có sử dụng hàm CVer, người dùng phải khai báo kiểu dữ liệu trả về của hàm là kiểu Variant.

Và như vậy, mỗi khi giá trị đầu vào không đúng, hàm sẽ trả về giá trị lỗi, giúp người sử dụng có thể nhận ra và sửa lỗi kịp thời.



Hình IV-9: Hàm trả về lỗi

Trong đoạn mã lệnh trên, để trả về thông báo lỗi “#N/A” thì tham số của hàm CVer phải là hằng số xlErrNA. Có rất nhiều giá trị lỗi khác nhau, mỗi giá trị lỗi có một hằng số tương ứng. Bảng dưới đây sẽ liệt kê một số giá trị lỗi cũng như các hằng số tương ứng trong VBA.

Giá trị lỗi	Hằng số	Giải thích
#DIV/0!	xlErrDiv0	Công thức có chia một số cho 0. Lỗi này cũng phát sinh khi chia cho một ô trống.
#N/A	xlErrNA	Lỗi này biểu thị dữ liệu không có.
#NAME?	xlErrName	Hàm có tên mà Excel không thể nhận dạng được. Thường xảy ra khi nhập tên hàm sai, hoặc đã thay đổi tên hàm nhưng chưa cập nhật trong bảng tính.
#NULL!	xlErrNull	Giá trị rỗng, chẳng hạn như tìm giao của hai vùng không giao nhau.
#NUM!	xlErrNum	Có vấn đề với giá trị nào đó. Ví dụ như người dùng nhập vào số âm, trong khi chỉ chấp nhận số dương.
#REF!	xlErrRef	Tham chiếu đến ô không tồn tại. Điều này thường xảy ra khi ô đã bị xóa khỏi bảng tính.
#VALUE!	xlErrValue	Hàm có chứa tham số hoặc công thức không phù hợp về kiểu dữ liệu

#### 4. Add-in và Phân phối các ứng dụng mở rộng

Một tính năng rất hữu ích cho người lập trình trong Excel là khả năng tạo Add-In. Phần này sẽ trình bày những lợi ích khi sử dụng Add-In, cách thức tạo và sử dụng Add-In.

## 4.1. Khái niệm về Add-In

Add-In là một chương trình gắn thêm vào Excel nhằm bổ sung thêm tính năng cho Excel. Thông thường, các tệp chứa Add-In có phần mở rộng là XLA và có cấu trúc tương tự như các workbook của Excel.

So với các ứng dụng trong tệp XLS của Excel, việc sử dụng Add-In có một số ưu điểm sau:

- ◆ Đơn giản hoá việc sử dụng hàm. Đối với các hàm lưu trong Add-In, khi sử dụng không cần phải thêm tên workbook ở phía trước tên hàm. Lấy ví dụ như người dùng tạo một hàm có tên là MOVAVG trong workbook có tên là Newfuncs.xls. Khi muốn sử dụng hàm đó trong một workbook khác, người dùng phải sử dụng hàm với cách thức như sau:

=Newfuncs.xls!MOVAVG(A1:A50)

Nhưng nếu hàm đó được lưu trong một Add-In đang được mở trong Excel, người dùng không cần phải thêm tên Add-In hay tên workbook ở trước tên hàm nữa, chỉ đơn giản là sử dụng tên hàm mà thôi:

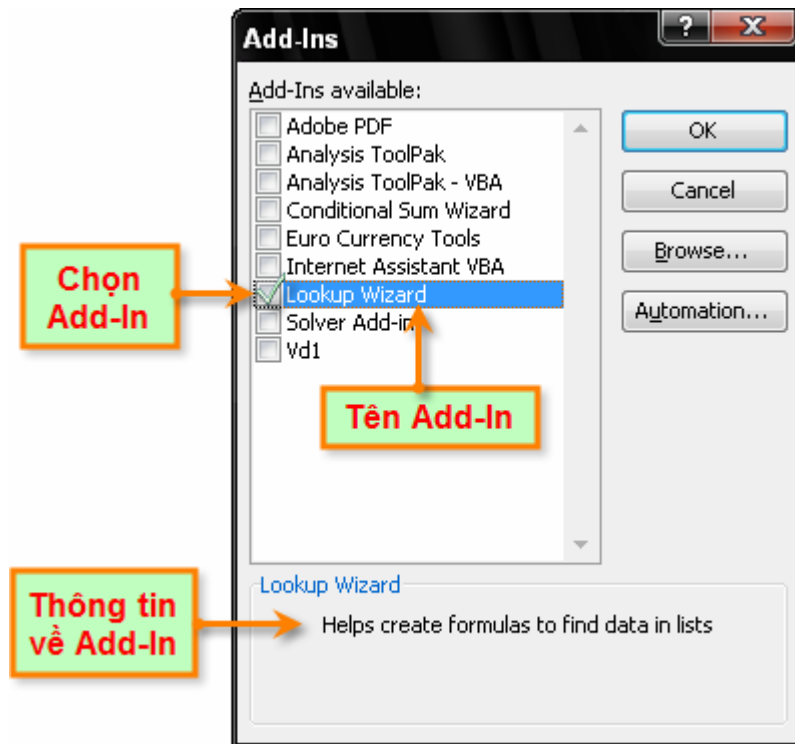
=MOVAVG(A1:A50)

Đơn giản hoá quá trình mở ứng dụng. Mỗi Add-In sau khi đã được cài đặt sẽ được tự động mở mỗi khi bắt đầu khởi động chương trình Excel. Hơn nữa, hộp thoại cảnh báo an ninh của Excel sẽ không xuất hiện (xem mục “Khi Macro được tạo ra chưa đáp ứng đủ nhu cầu thì người sử dụng có thể thay đổi, bổ sung mã lệnh cho Macro đó. Quá trình hiệu chỉnh Macro được thực hiện thông qua VBAIDE. Để hiệu chỉnh Macro, ta có thể dùng trình quản lý Macro (xem mục “*Quản lý Macro*” trang 104) hoặc truy cập trực tiếp trong VBAIDE. Về bản chất, việc hiệu chỉnh (sửa đổi) Macro tương đương như việc lập trình để xây dựng nên Macro đó.

- ◆ Vấn đề an toàn khi sử dụng Macro” trang 107), tránh gây ra sự lúng túng cho những người dùng chưa có kinh nghiệm.
- ◆ Tránh gây ra sự bối rối cho người dùng bởi toàn bộ dữ liệu trong các Sheet của tệp Add-In được che dấu, như vậy, với người dùng ít kinh nghiệm, họ sẽ không phải thắc mắc hay cảm thấy khó hiểu khi không nhìn thấy những dữ liệu này.
- ◆ Ngăn chặn việc truy cập vào mã lệnh. Khi phân phối ứng dụng dạng Add-In có đặt chế độ bảo mật bằng mật khẩu, người dùng không thể xem hoặc thay đổi mã lệnh của ứng dụng. Điều này tránh được việc sao chép mã lệnh của chương trình.

## 4.2. Trình quản lý Add-In

Việc quản lý các Add-In trong Excel được thực hiện rất đơn giản thông qua trình quản lý Add-In. Để hiển thị trình quản lý Add-In, chọn trình đơn **Tools**⇒**Add-Ins...**



Hình IV-10: Trình quản lý Add-In

- ◆ Để tải/dỡ bỏ Add-In trong Excel: kích chuột vào hộp kiểm ở bên trái tên của Add-In.
- ◆ Để mở một Add-In: chọn nút lệnh **Browse...** ⇒ Chọn Add-In cần mở.



**GỢI Ý** Thông thường, tệp Add-In sẽ có phần mở rộng là XLA và được lưu trữ trong thư mục %UserProfile%\Application Data\Microsoft\AddIns.

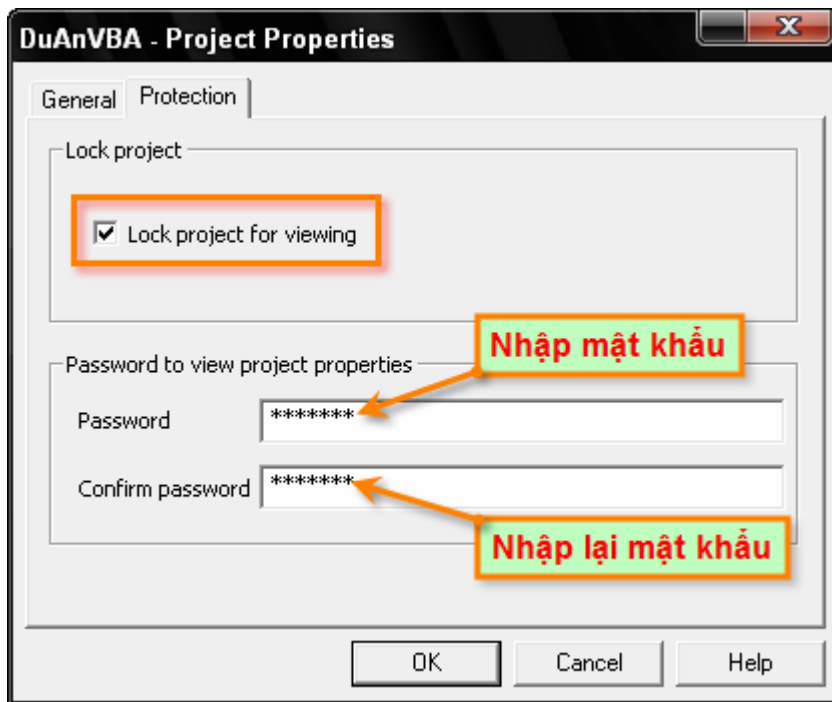
### 4.3. Tạo Add-In

Nhìn chung, việc tạo Add-In được thực hiện rất dễ dàng bằng cách chuyển từ workbook thông thường sang dạng Add-In. Đương nhiên, không phải workbook nào cũng thích hợp để chuyển thành Add-In. Nhìn chung, workbook thích hợp nhất để chuyển thành Add-In là workbook có chứa mã lệnh. Một workbook nếu chỉ chứa worksheet thì có thể sẽ không thể sử dụng được khi chuyển thành Add-In, bởi lẽ tất cả các sheet trong workbook sẽ bị ẩn đi khi được chuyển thành Add-In.

Việc tạo Add-In từ một workbook thông thường được thực hiện rất dễ dàng theo các bước sau:

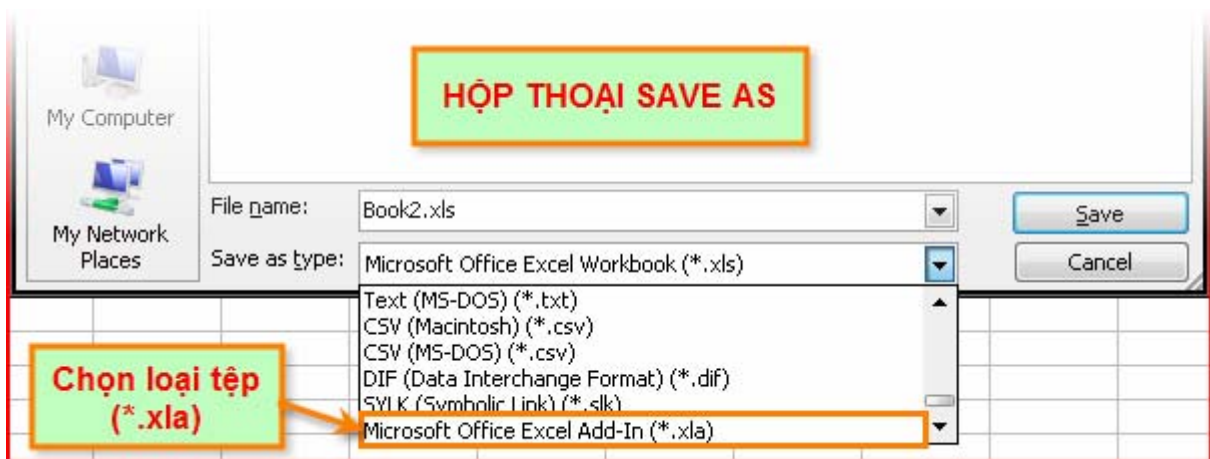
1. Viết mã lệnh cho workbook như bình thường, đảm bảo tất cả các mã lệnh đều có thể thực hiện bình thường mà không có lỗi xảy ra. Nên nhớ là tạo giao diện sao cho người dùng có thể truy cập và sử dụng được các tính năng trong Add-In. Một cách để thực hiện việc này là tạo một trình đơn mới cho Add-In, trong đây có chứa các mục trình đơn tương ứng với các tính năng của ứng dụng. Chi tiết về cách tạo trình đơn, xem mục “*Tạo trình đơn tùy biến*” trang 174.
2. Kiểm tra ứng dụng bằng cách thực thi khi đang kích hoạt một workbook khác. Điều này mô phỏng được tình huống như khi workbook đã được chuyển thành Add-In, bởi lẽ bản thân Add-In không hiển thị các sheet mà lúc này tất cả các thao tác đều thực hiện trên các sheet của workbook khác.
3. Nếu không muốn người khác xem được mã lệnh của ứng dụng thì thực hiện bước này: đặt mật khẩu bảo vệ. Kích hoạt VBAIDE ⇒ chọn workbook tương ứng trong cửa sổ Project. Chọn trình đơn **Tools** ⇒ **xxx Properties...** (xxx là tên Dự án VBA), sau đó chọn thẻ

**Protection.** Tích vào ô **Lock project for viewing**, và nhập vào mật khẩu (2 lần) ⇒ chọn OK.



**Hình IV-11: Hộp thoại Project Properties**

4. Kích hoạt lại Excel, chọn trình đơn **File** ⇒ **Properties**, sau đó chọn thẻ **Summary**. Nhập vào các thông tin cho Add-In để hiển thị trong trình quản lý Add-In. Mục **Title** sẽ là tên của Add-In, nếu không nhập vào mục này, tên của Add-In sẽ là tên tệp Add-In. Mục **Comments** sẽ là phần mô tả về Add-In.
5. Chọn trình đơn **File** ⇒ **Save As...**
6. Trong hộp thoại **Save As**, chọn **Microsoft Excel add-in (\*.xla)** trong mục **Save as type**.



**Hình IV-12: Hộp thoại Save As**

7. Chọn **Save**. Một bản sao của workbook đã được lưu (với phần mở rộng là \*.xla), và tệp XLS chứa workbook vẫn còn giữ nguyên trong Excel.



**CHÚ Ý** Một workbook khi được chuyển thành Add-In phải có ít nhất một worksheet. Chẳng hạn như khi một workbook chỉ chứa Chart Sheet hoặc Dialog Sheet, thì lựa chọn Microsoft Excel add-in (\*.xla) sẽ không xuất hiện trong mục Save as type trong hộp thoại Save As. Lựa chọn này chỉ xuất hiện khi có một worksheet được chọn lúc chọn trình đơn **File** ⇒ **Save As**.

Sau khi đã tạo Add-In, nên lưu giữ lại workbook nguồn (dạng XLS) để có thể hiệu chỉnh hay cập nhật mã lệnh và các dữ liệu khác sau này. Cần phải làm điều này vì tệp Add-In không thể chuyển đổi ngược lại thành workbook.

### 4.4. Phân phối và Cài đặt Add-In

Việc phân phối các Add-In được thực hiện rất đơn giản, chỉ cần sao chép tệp \*.xla đến các máy khác, sau đó cài đặt các Add-In thông qua trình quản lý Add-In trong Excel.

Để cài đặt Add-In, thực hiện theo các bước sau:

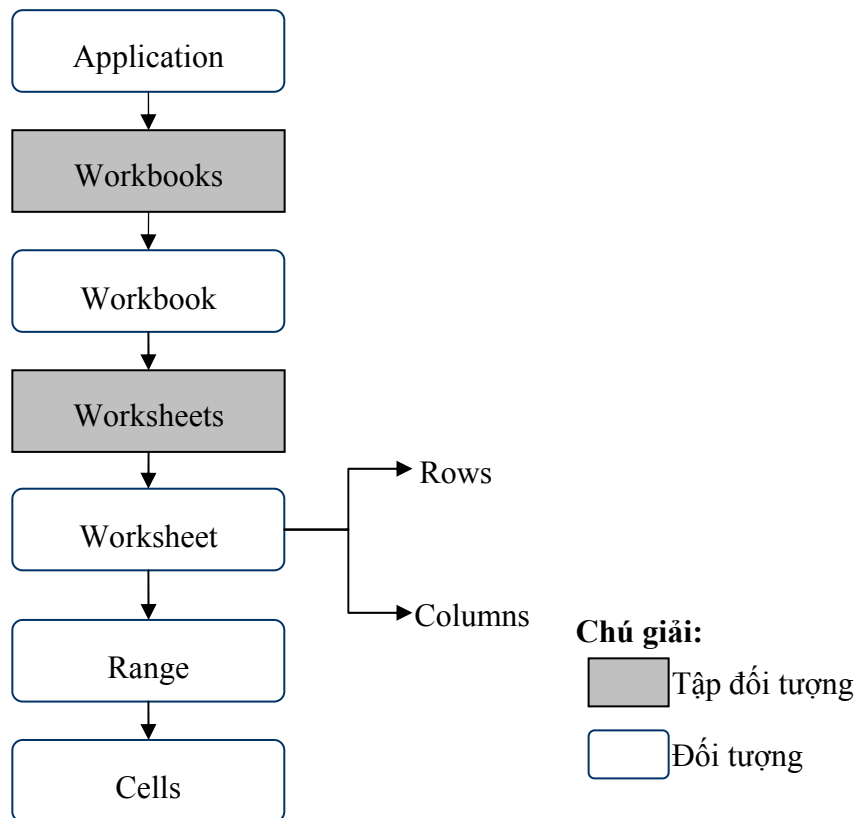
1. Trong Excel, chọn trình đơn **Tools** ⇒ **Add-Ins...** để hiển thị trình quản lý Add-In
2. Chọn nút **Browse**, sau đó trở đến tệp Add-In cần cài đặt trong Excel ⇒ chọn **OK**. Tên của Add-In sẽ được hiển thị trong trình quản lý Add-In.
3. Chọn **OK** lần nữa để chấp nhận cài đặt Add-In. Giờ đây, người dùng có thể sử dụng tất cả các tính năng có trong Add-In vừa được cài đặt.

## 5. Hệ thống các đối tượng trong Excel

Điểm khác biệt của lập trình trên Excel so với việc lập trình trên các ứng dụng nền khác chính là việc thực hiện các thao tác nhằm tác động trực tiếp đến các thành phần trong Excel thông qua công cụ lập trình. Vì vậy, để có thể tạo ra các ứng dụng trên nền Excel, người dùng cần phải hiểu rõ thành phần cũng như cách thao tác trên các thành phần đó của Excel.

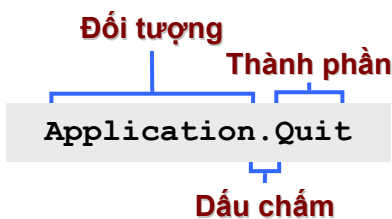
### 5.1. Mô hình đối tượng trong Excel

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. Mô hình đối tượng đầy đủ được trình bày trong tài liệu hướng dẫn của Excel hoặc trong các tài liệu tham khảo ở cuối giáo trình này. Ở đây chỉ đề cập đến một số đối tượng thường được sử dụng trong lập trình trên Excel.



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). Chẳng hạn như đối tượng `Application`, là đối tượng thể hiện cho Excel, có thuộc tính `Caption` chứa tiêu đề của Excel và phương thức `Quit` dùng để thoát khỏi Excel.

Để truy cập đến các thành phần (phương thức, thuộc tính, ...) của đối tượng, ta sử dụng dấu chấm (.), ví dụ sau thực hiện phương thức `Quit` để thoát khỏi Excel như đã đề cập ở trên:



### Cấu trúc phân cấp đối tượng

Đối tượng `Application` (chính là ứng dụng Excel) chứa nhiều đối tượng khác, chẳng hạn như:

- ◆ `Workbooks` (tập đối tượng chứa tất cả các đối tượng `Workbook` – tài liệu Excel)
- ◆ `Windows` (tập đối tượng chứa tất cả các đối tượng `Window` - các cửa sổ trong Excel)
- ◆ `AddIns` (tập đối tượng chứa tất cả các đối tượng `Add-in`)

Tập đối tượng `Workbooks` chứa tất cả các đối tượng `Workbook` đang mở, và mỗi đối tượng `Workbook` lại chứa các đối tượng khác như:

- ◆ `Worksheets` (tập đối tượng chứa các đối tượng `Worksheet`)
- ◆ `Charts` (tập đối tượng chứa các đối tượng `Chart`)

Đến lượt mình, các đối tượng trên cũng có thể chứa nhiều đối tượng khác nữa. Một đối tượng `Worksheet` trong tập đối tượng `Worksheets` có thể chứa các đối tượng khác, chẳng hạn như:

- ◆ ChartObjects (tập đối tượng chứa tất cả đối tượng ChartObject – biểu đồ trong Excel)
- ◆ Range
- ◆ PageSetup

Cứ như vậy, người lập trình có thể truy cập đến từng thành phần của Excel thông qua hệ thống phân cấp các đối tượng trong Excel.

### Tập đối tượng – Collection

Một khái niệm rất quan trọng trong lập trình VBA là khái niệm tập đối tượng (hay 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). Chẳng hạn 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:

```
Worksheets("Sheet1")
```

Nếu Sheet1 là sheet đầu tiên trong tập đối tượng Worksheet, thì ta còn có thể tham chiếu dựa trên số thứ tự của Sheet1 theo cách sau:

```
Worksheets(1)
```

Tương tự, để tham chiếu đến Sheet thứ 2 trong tập đối tượng Worksheets, ta có thể sử dụng lệnh: Worksheets(2).

Mỗi tập đối tượng có các phương thức dùng để thao tác trên chính tập đối tượng đó. Các phương thức này rất khác nhau trên các tập đối tượng khác nhau. Vì vậy, người lập trình có thể sử dụng Object Browser để tìm hiểu về các phương thức trong tập đối tượng.

## 5.2. Một số đối tượng cơ bản trong Excel

Hệ thống đối tượng trong Excel rất đa dạng và tương ứng dùng để biểu diễn các thành phần trong Excel. Phần này sẽ giới thiệu chi tiết về các đối tượng và tập đối tượng chính trong Excel cùng với các phương thức/thuộc tính của chúng thông qua các mã lệnh tương ứng.

### 5.2.1. Đối tượng Application

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.

Đối tượng Application có chứa nhiều thiết lập cho ứng dụng (chẳng hạn như các lựa chọn trong trình đơn **Tools⇒Options...**) và rất nhiều đối tượng trong ứng dụng (chẳng hạn như các tài liệu đang được mở - Workbooks, hay bảng tính hiện hành -ActiveSheet...).

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. Đoạn mã lệnh sau sẽ khởi động Excel từ chương trình ngoài (ví dụ như khi 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
```

## GIÁO TRÌNH TỰ ĐỘNG HOÁ THIẾT KẾ CẦU ĐƯỜNG

Dưới đây là các phương thức và thuộc tính thường được sử dụng trong đối tượng Application.

### ActiveCell

Thuộc tính ActiveCell thể hiện cho ô hiện thành trong bảng tính Excel. Kiểu dữ liệu của ActiveCell là kiểu Range. Người dùng có thể truy cập đến địa chỉ của ô hiện hành bằng cách truy cập thêm vào một cấp nữa trong cây phân cấp đối tượng, đó là thuộc tính Address:

```
MsgBox Application.ActiveCell.Address
```

Đoạn mã trên hiển thị địa chỉ của ô hiện hành theo dạng địa chỉ tuyệt đối, chẳng hạn như \$A\$6. Cần lưu ý là thuộc tính Address chỉ trả về địa chỉ của ô, chứ không phải địa chỉ đầy đủ bao gồm cả tên sheet và workbook.

### ActivePrinter

Thuộc tính ActivePrinter chứa tên của máy in hiện hành. Kiểu dữ liệu của thuộc tính ActivePrinter là kiểu String. Đoạn mã sau hiển thị tên của máy in hiện hành:

```
MsgBox Application.ActivePrinter
```

Thuộc tính này rất có ích khi thông báo cho người dùng biết về máy in hiện hành trước khi in một bảng tính nào đó.

### 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..., chi tiết xem thêm mục “*Đối tượng Workbook*” trang 123. Đoạn mã sau sử dụng hàm TypeName, hàm trả về kiểu dữ liệu của biến, để từ đó biết được kiểu của sheet hiện hành:

```
MsgBox TypeName(Application.ActiveSheet)
```

### ActiveWindow

Thuộc tính này trả về đối tượng chứa cửa sổ hiện hành, nếu không cửa sổ nào được mở thì sẽ trả về giá trị Nothing. Kiểu dữ liệu của thuộc tính này là Window. Đoạn mã sau sẽ thu 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 trả về giá trị Nothing. Kiểu dữ liệu của thuộc tính này là Workbook. 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 sổ mới bằng cách chọn trình đơn **Window**⇒**New Window**. Cửa sổ mới được tạo có nội dung giống như cửa sổ ban đầu, nhưng người dùng có thể lựa chọn những vùng khác nhau trên hai cửa sổ (mặc dù cả hai cửa sổ đều là



thể hiện của cùng một workbook). Và như vậy, mỗi một workbook có thể được hiển thị trên nhiều cửa sổ khác nhau, nhưng mỗi một cửa sổ chỉ hiển thị được một workbook mà thôi.

### AddIns

Là tập đối tượng chứa tất cả các add-in đã được tải vào trong Excel. Đoạn mã lệnh sau sẽ lần lượt hiển thị tên của tất cả các add-in, bao gồm cả đường dẫn:

```
Sub Hien_thi_Add_in()  
    Dim MyAddin As AddIn  
    For Each MyAddin In Application.AddIns  
        MsgBox MyAddin.FullName  
    Next MyAddin  
End Sub
```

### Calculate

Là phương thức thực hiện quá trình tính toán lại trên toàn bộ sheet, giống như khi nhấn phím F9.

```
Application.Calculate
```

### Calculation

Là thuộc tính dùng để thiết lập chế độ thực hiện tính toán trong chương trình Excel. Các giá trị có thể gán cho thuộc tính này là xlCalculationAutomatic (tính tự động), xlCalculationManual (tính thủ công) và xlCalculationSemiautomatic (tính bán tự động – tính tự động ngoại trừ phần bảng). Ví dụ đoạn mã sau chuyển chế độ tính thành bán tự động:

```
Application.Calculation= xlCalculationSemiautomatic
```

Điều này tương đương với việc chọn trình đơn **Tools⇒Options**, chọn thẻ Calculation và chọn lựa chọn Automatic except tables.

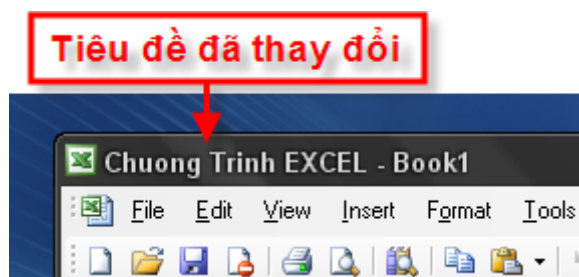
Chế độ tính mặc định trong Excel là chế độ tính tự động. Tuy nhiên, đối với những bảng tính có khối lượng tính toán lớn, nếu để chế độ tính tự động thì mỗi lần thay đổi số liệu là một lần thực hiện tính toán. Trong những trường hợp như vậy, nên chuyển sang chế độ thủ công, sau đó tiến hành thay đổi số liệu tính toán. Sau khi đã thay đổi xong các số liệu thì mới tiến hành tính toán một lần cuối.

### Caption

Là thuộc tính dùng để chứa tiêu đề của chương trình Excel. Kiểu dữ liệu của thuộc tính này là String. Đoạn mã sau sẽ thay đổi tiêu đề của chương trình Excel:

```
Application.Caption = "Chuong Trinh EXCEL"
```

Và đây là kết quả thực thi đoạn mã trên



## GIÁO TRÌNH TỰ ĐỘNG HOÁ THIẾT KẾ CẦU ĐƯỜNG

Muốn thiết lập lại giá trị mặc định (Microsoft Excel), chỉ cần gán giá trị của `Caption= ""`

### Columns và Rows

Là hai tập đối tượng thể hiện tất cả các cột và các hàng trong sheet hiện hành, ta có thể sử dụng để truy cập một cột hoặc một hàng nào đó.

Đoạn mã sau sẽ chọn toàn bộ cột C:

```
Application.Columns(3).Select
```

Còn đoạn mã sau sẽ chọn toàn bộ hàng thứ 9:

```
Application.Rows(9).Select
```

### Dialogs

Dialogs là tập đối tượng chứa tất cả các hộp thoại đã được định nghĩa sẵn trong Excel. Chi tiết về tập đối tượng này, tham khảo mục “*Các hộp thoại mặc định trong Excel – Tập đối tượng Dialogs*” trang 166.

### Help

Phương thức này hiển thị tệp trợ giúp do người dùng chỉ định.

```
Application.Help "C:\Program Files\" & _  
"Microsoft Office\OFFICE11\1033\VBAXL10.CHM"
```

Nếu không chỉ ra tệp trợ giúp, phương thức này sẽ hiển thị tệp trợ giúp mặc định trong Excel.

### Quit

Phương thức này sẽ đóng chương trình Excel lại, giống như khi lựa chọn trình đơn **File⇒Exit**. Chương trình sẽ nhắc người dùng lưu lại các tệp chưa được lưu.

```
Application.Quit
```

### RecentFiles

RecentFiles là tập đối tượng lưu giữ những tệp mở sau cùng nhất trong Excel. Mỗi đối tượng trong tập đối tượng RecentFiles có kiểu dữ liệu là RecentFile. Ví dụ sau sẽ hiển thị lần lượt tên của các tệp mở sau cùng nhất trong Excel:

```
Public Sub RecentFile()  
    Dim myRecentFile As RecentFile  
    For Each myRecentFile In Application.RecentFiles  
        MsgBox myRecentFile.Path  
    Next myRecentFile  
End Sub
```

### Selection

Thuộc tính này thể hiện cho đối tượng đang được chọn trong Excel. Kiểu dữ liệu trả về của thuộc tính này tùy thuộc vào đối tượng được chọn. Nếu đối tượng chọn là các ô, biểu đồ, hoặc một đường thẳng thì kiểu dữ liệu trả về tương ứng sẽ là Range, ChartArea, Line... Vì vậy khi lập trình, cần phải chú ý kiểm tra kiểu dữ liệu trả về của thuộc tính này để có các thao tác hợp lý, tránh lỗi xảy ra khi thực thi chương trình. Ví dụ sau sẽ hiển thị địa chỉ của các ô đang được chọn trong Excel:

```
Dim mySelection As Variant
Set mySelection = Selection
If TypeName(mySelection) = "Range" Then           'Kiểm tra kiểu dữ
liệu
    MsgBox mySelection.Address
Else
    MsgBox "Đối tượng được chọn không phải kiểu Range"
End If
```

## Sheets

Sheets là tập đối tượng chứa tất cả các sheet có trong workbook hiện hành, gồm cả 4 loại sheet: worksheet, chart sheet, macro sheet và dialog sheet. Đoạn macro sau sẽ hiển thị tên và kiểu của tất cả các sheet có trong workbook hiện hành:

```
Sub Sheets()
    Dim mySheet As Variant
    For Each mySheet In Application.Sheets
        MsgBox mySheet.Name & " - " & TypeName(mySheet)
    Next mySheet
End Sub
```

## ThisWorkbook

Thuộc tính này trả về đối tượng thể hiện cho workbook hiện hành, nơi đang thực hiện macro. Kiểu dữ liệu của thuộc tính này là Workbook.

## Undo

Phương thức này sẽ khôi phục lại các thao tác trước đó trong Excel. Thao tác này tương tự như khi chọn trình đơn **Edit⇒Undo...** trong Excel.

```
Application.Undo
```



**CHÚ Ý** Khi truy cập đến một đối tượng thuộc một đối tượng khác, nếu không chỉ rõ tham chiếu đối tượng thì đối tượng được tham chiếu sẽ là đối tượng hiện hành. Ví dụ như nếu muốn truy cập vào ô A1 của Sheet1 của workbook hiện hành, thay vì sử dụng câu lệnh "ActiveWorkbook.Worksheets("Sheet1").Range("A1")" chỉ cần dùng câu lệnh "Worksheets("Sheet1").Range("A1")". Hoặc nếu sử dụng câu lệnh "Range("A1").Select" thì ô A1 của sheet hiện hành sẽ được chọn.

### 5.2.2. Đối tượng Workbook

Workbook là một đối tượng phổ biến trong Excel. Hầu hết tất cả các thao tác trên Excel đều được thực hiện trên một workbook nào đó. Trong mỗi phiên làm việc của Excel có thể có rất nhiều workbook được mở, và việc truy cập đến một workbook nào đó sẽ được thực hiện thông qua tập đối tượng Workbooks, dựa trên tên hoặc chỉ số của các workbook, chỉ số này phụ thuộc vào trình tự mở/tạo workbook. Ngoài ra, người lập trình còn có thể truy xuất đến workbook hiện hành thông qua đối tượng ActiveWorkbook.

Để tạo mới một workbook, sử dụng phương thức Add có trong tập đối tượng Workbooks:

```
Workbooks.Add           'Tạo mới một Workbook
```

Mỗi workbook trong Excel có thể chứa nhiều sheet khác nhau (tùy thuộc vào dung lượng bộ nhớ). Có 4 loại sheet khác nhau:

- ✦ **Worksheet:** đây là loại sheet phổ biến nhất, là loại sheet thực hiện việc tính toán và thao tác chính trong Excel. Mỗi sheet bao gồm 256 cột và 65536 hàng tạo thành hệ thống các ô chứa dữ liệu. Ngoài ra, trên mỗi worksheet còn thể chứa các đối tượng khác như biểu đồ, các đối tượng đồ hoạ, các điều khiển,...
- ✦ **Chart sheet:** mỗi chart sheet thường chỉ chứa một biểu đồ. Thông thường, người dùng thích sử dụng biểu đồ nhúng trong worksheet hơn là sử dụng biểu đồ trong một chart sheet riêng biệt. Tuy nhiên, chart sheet lại giúp cho việc bố trí và in ấn biểu đồ thực hiện dễ dàng hơn
- ✦ **XLM macro sheets** (còn gọi là MS Excel 4 macro sheet, là loại sheet đã lỗi thời, nhưng vẫn còn hỗ trợ): dùng để chứa các macro XLM. XLM macro sheet cũng có chứa các ô dữ liệu nhưng chỉ hiển thị công thức chứ không hiển thị kết quả tính.
- ✦ **Dialog sheets** (đã lỗi thời, nhưng vẫn còn hỗ trợ): là nơi chứa các hộp thoại tùy biến do người dùng tự tạo (giống như Userform trong VBA).

Các phương thức và thuộc tính thường được sử dụng của đối tượng workbook:

### Activate

Phương thức này sẽ kích hoạt một workbook trong tập đối tượng Workbooks thành workbook hiện hành, và sheet hiện hành của workbook đó sẽ được kích hoạt làm sheet hiện hành trong Excel.

Để kích hoạt workbook có tên Book1 làm workbook hiện hành, thực hiện như sau:

```
Workbooks("Book1").Activate
```

Còn để kích hoạt workbook đầu tiên trong Excel, thực hiện như sau:

```
Workbooks(1).Activate
```

### ActiveSheet

Thuộc tính này tham chiếu đến sheet hiện hành của workbook. Trong mỗi phiên làm việc của Excel, số lượng Workbook được mở là không hạn chế, mỗi workbook này đều có một sheet hiện hành, nghĩa là khi ta chọn làm việc với workbook đó thì con trỏ sẽ nằm trong sheet này. Kiểu dữ liệu trả về của thuộc tính này tùy thuộc vào kiểu sheet hiện hành (một trong 4 loại). Trong đối tượng Application cũng có thuộc tính ActiveSheet, nhưng thuộc tính này là sheet hiện hành của workbook hiện hành, nghĩa là nơi có con trỏ đang hoạt động, như vậy, nếu workbook này là hiện hành thì thuộc tính ActiveSheet của đối tượng Application và của đối tượng Workbook là như nhau.

Đoạn mã lệnh sau hiển thị tên của sheet hiện hành của workbook tên là Book1:

```
MsgBox Workbooks("Book1").ActiveSheet.Name
```

### Close

Phương thức này sẽ đóng workbook lại, tương tự như khi sử dụng trình đơn **File⇒Close** trong Excel. Ngoài ra còn có các tham số tùy chọn khác phục vụ cho việc lưu trữ tệp.

```
Workbooks("Book1").Close ([SaveChanges], [Filename])
```

Trong đó ý nghĩa của các tham số như sau:

- ◆ `SaveChanges`: tham số tùy chọn. Bằng `TRUE` nếu muốn lưu tất cả các thay đổi, và bằng `FALSE` nếu chỉ muốn đóng workbook mà không lưu. Nếu bỏ qua, tùy chọn này thì phương thức này sẽ giống hoàn toàn như khi ta chọn trình đơn **File⇒Close**.
- ◆ `Filename`: tham số tùy chọn. Sẽ lưu tệp với tên chứa trong `Filename`.

### PrintOut

Phương thức này sẽ in sheet hiện hành của workbook ra máy in.

```
Workbooks(1).PrintOut
```

Phương thức này còn có rất nhiều tham số khác nữa, tất cả đều là tham số tùy chọn:

- ◆ `From`: số thứ tự trang bắt đầu in, nếu bỏ qua thì sẽ in từ đầu.
- ◆ `To`: số thứ tự trang cuối cùng được in, nếu bỏ qua thì sẽ in đến trang cuối cùng.
- ◆ `Copies`: số bản sao khi in ra, nếu bỏ qua thì chỉ in một bản.
- ◆ `Preview`: nếu bằng `TRUE`, Excel sẽ hiển thị cửa sổ xem trước khi in. Nếu bằng `FALSE`, hoặc bỏ qua, thì sẽ in trực tiếp.
- ◆ `ActivePrinter`: thiết lập tên cho máy in hiện hành.
- ◆ `PrintToFile`: nếu bằng `TRUE` sẽ in ra tệp. Trong trường hợp đó, nếu không gán giá trị cho tham số `PrToFileName`, Excel sẽ hiển thị hộp thoại để người dùng nhập vào tên tệp.
- ◆ `Collate`: nếu bằng `TRUE` và số bản sao lớn hơn 1, Excel sẽ sắp xếp các bản in thành từng tập hoàn thiện.
- ◆ `PrToFileName`: nếu tham số `PrintToFile` gán bằng `TRUE` thì tham số này sẽ thiết lập tên tệp để in ra.

### PrintPreview

Phương thức này sẽ hiển thị chế độ xem trước khi in cho sheet hiện hành của workbook tham chiếu.

```
Workbooks(1).PrintPreview
```

### Save và SaveAs

Các phương thức này sẽ lưu workbook và thường được dùng trước khi đóng workbook. Phương thức `Save` sẽ lưu những thay đổi vào chính workbook đó. Còn phương thức `SaveAs` sẽ lưu workbook ra một tệp mới. Đoạn mã sau sẽ lưu workbook có tên `Book1` và sau đó lưu workbook có tên là `Book3` với tên mới là `MyFile.xls`:

```
Workbooks("Book1").Save  
Workbooks("Book3").SaveAs "C:\MyFile.xls"
```

### Saved

Thuộc tính này trả về giá trị `TRUE` nếu workbook đã được lưu, và ngược lại là `FALSE`. Đoạn mã sau sẽ hiển thị trạng thái lưu của workbook:

```
MsgBox Workbooks(1).Saved
```

### Sheets

Tập đối tượng Sheets của workbook cũng giống như tập đối tượng Sheets của đối tượng Application (xem lại mục “*Sheets*” trang 123). Tuy nhiên, tập đối tượng sheets của workbook tham chiếu trực tiếp đến các sheet trong workbook đó, còn tập đối tượng sheets của đối tượng Application lại tham chiếu đến các sheet trong workbook hiện hành.

### Windows

Windows là tập đối tượng chứa tất cả các cửa sổ có trong đối tượng Workbook. Chi tiết về tập đối tượng Windows, xem thêm mục “*Đối tượng Window*” trang 126.

### Worksheets

Worksheets là tập đối tượng chứa tất cả các worksheet có trong đối tượng Workbook. Chi tiết về tập đối tượng Worksheets, xem thêm mục “*Đối tượng Worksheet*” trang 128.

#### 5.2.3. Đối tượng Window

Đối tượng Window thể hiện cho một cửa sổ bên trong ứng dụng Excel. Như đã được đề cập, đối tượng Window rất dễ nhầm lẫn với đối tượng Workbook. 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 đối tượng workbook và đối tượng window, ta tạo thêm một cửa sổ mới bằng cách chọn trình đơn **Window**⇒**New Window**. Cửa sổ mới được tạo có nội dung giống như cửa sổ ban đầu, nhưng người dùng có thể làm việc trên hai cửa sổ này giống như khi làm việc trên 2 workbook riêng biệt (mặc dù cả hai cửa sổ đều là thể hiện của cùng một workbook). Đương nhiên, sự thay đổi ở cửa sổ này sẽ được tự động cập nhập trong các cửa sổ còn lại. Và như vậy, mỗi một workbook có thể được thể hiện bằng nhiều cửa sổ khác nhau, nhưng mỗi một cửa sổ chỉ thể hiện được một workbook mà thôi.

Để truy xuất đến một đối tượng trong tập đối tượng Windows, người dùng có thể truy cập theo tên hoặc theo thứ tự của cửa sổ. Để truy xuất đến cửa sổ có tiêu đề là Book1 (là dòng chữ xuất hiện trên thanh tiêu đề của cửa sổ), ta sử dụng cấu trúc sau:

```
Windows ("Book1")
```

Để truy xuất đến cửa sổ thứ 2 trong tập đối tượng Windows, ta sử dụng cấu trúc sau:

```
Windows (2)
```

Số thứ tự của một cửa sổ là không cố định mà thay đổi tùy theo số cửa sổ hiển thị và sự thay đổi của cửa sổ hiện hành. Cửa sổ hiện hành luôn có thứ tự là 1.

Dưới đây là các phương thức và thuộc tính thường sử dụng trong đối tượng Window:

#### Activate, ActivateNext và ActivatePrevious

Các phương thức này dùng để kích hoạt một cửa sổ nào đó trong tập đối tượng Windows. Activate, ActivateNext và ActivatePrevious dùng để kích hoạt cửa sổ được chỉ định, cửa sổ tiếp theo và cửa sổ trước của cửa sổ được chỉ định. Đoạn mã sau sẽ kích hoạt cửa sổ tên là Book1, sau đó sẽ kích hoạt cửa sổ nằm phía trước của cửa sổ Book1.

```
Windows ("Book1").Activate  
Windows ("Book1").ActivatePrevious
```

#### ActiveCell

Thuộc tính này tham chiếu đến ô hiện hành, là ô có con trỏ đang hoạt động, trong đối tượng Window. Đoạn mã sau sẽ hiển thị địa chỉ của ô hiện hành trong cửa sổ thứ 2 trong tập đối tượng Windows:

```
MsgBox Windows (2) .ActiveCell.Address
```

### ActiveSheet

Thuộc tính này tham chiếu đến sheet hiện hành của workbook đang xét.

### Caption

Thuộc tính này chứa nội dung ghi trên thanh tiêu đề của cửa sổ. Đoạn mã sau sẽ thay đổi dòng tiêu đề của cửa sổ hiện hành:

```
ActiveWindow.Caption = "MyWindow"
```

### Close

Close là hàm thực hiện đóng cửa sổ đang xét. Hàm này trả về giá trị TRUE nếu đóng được cửa sổ, và trả về giá trị FALSE nếu cửa sổ không được đóng. Hàm này cũng chứa các tham số tùy chọn khác, chi tiết xem thêm phương thức Close của đối tượng Workbook trang 124.

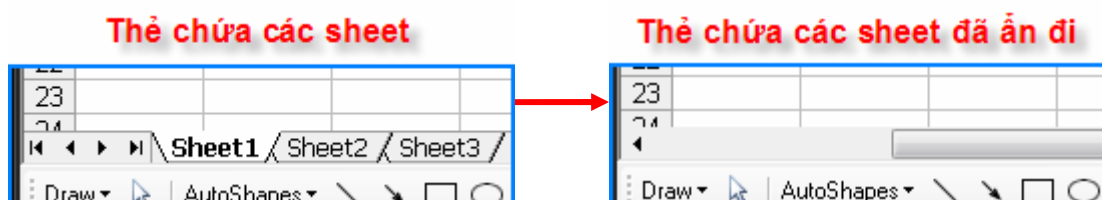
### Các thuộc tính về hiển thị

Đối tượng Window có chứa nhiều thuộc tính để thiết lập các lựa chọn về hiển thị trong cửa sổ như sau:

Thuộc tính	Giải thích
DisplayFormulas	TRUE: tất cả các ô sẽ hiển thị công thức chứ không phải giá trị
DisplayGridlines	TRUE: hiển thị các đường lưới bên trong cửa sổ
DisplayHeadings	TRUE: hiển thị thanh thể hiện vị trí của cột và hàng
DisplayHorizontalScrollBar	TRUE: hiển thị thanh cuộn ngang
DisplayOutline	TRUE: hiển thị thanh thể hiện Outline
DisplayRightToLeft	TRUE: hiển thị trật tự cột tăng dần từ phải sang trái. Mặc định là FALSE
DisplayVerticalScrollBar	TRUE: hiển thị thanh cuộn đứng
DisplayWorkbookTabs	TRUE: hiển thị thẻ chứa các sheet trong workbook
DisplayZeros	FALSE: các ô có giá trị bằng 0 sẽ không hiển thị

Đoạn mã sau sẽ ẩn đi thẻ chứa các sheet trong workbook hiện hành:

```
ActiveWindow.DisplayWorkbookTabs = False
```



### NewWindow

Phương thức này sẽ tạo một cửa sổ mới dựa trên cửa sổ đang được tham chiếu giống như khi chọn trình đơn **Window** ⇒ **New Window** trong Excel vậy.

```
ActiveWindow.NewWindow
```

### RangeSelection

## GIÁO TRÌNH TỰ ĐỘNG HOÁ THIẾT KẾ CẦU ĐƯỜNG

Thuộc tính này tham chiếu đến vùng đang được chọn trong cửa sổ được tham chiếu. Kiểu dữ liệu của thuộc tính này là Range. Đoạn mã sau sẽ hiển thị địa chỉ của vùng được chọn trong cửa sổ hiện hành:

```
MsgBox ActiveWindow.RangeSelection.Address
```

Giá trị hiển thị có thể ở dạng địa chỉ tuyệt đối của một ô - \$C\$3, hoặc một vùng chọn - \$B\$10:\$D\$12.

### SelectedSheets

SelectedSheets là tập đối tượng chứa tất cả các sheet đang được người dùng lựa chọn. Kiểu dữ liệu của thuộc tính này là Sheets, vì thế tất cả các thao tác trên tập đối tượng SelectedSheets cũng giống như trên tập đối tượng Sheets

### WindowState

Thuộc tính này lưu trữ trạng thái của đối tượng Window. Có 3 trạng thái của đối tượng Window như sau:

Trạng thái đối tượng Window	Giá trị của thuộc tính
Cửa sổ được phóng đại toàn màn hình	xlMaximized
Cửa sổ được thu nhỏ	xlMinimized
Cửa sổ ở trạng thái thông thường	xlNormal

Đoạn mã sau sẽ thu nhỏ cửa sổ hiện hành:

```
ActiveWindow.WindowState = xlMinimized
```

### Zoom

Thuộc tính này dùng để thiết lập chế độ phóng đại cho cửa sổ, giống như khi ta sử dụng trình đơn **View**⇒**Zoom** trong Excel. Đoạn mã sau sẽ phóng đại cửa sổ hiện hành lên 120%:

```
ActiveWindow.Zoom = 120
```



**GỢI Ý** Trong Excel, để thu/phóng cửa sổ, chỉ cần nhấn phím CTRL + cuộn phím giữa của chuột.

### 5.2.4. Đối tượng Worksheet

Đối tượng Worksheet thể hiện cho một worksheet trong một workbook. Đây là đối tượng rất phổ biến trong excel, hầu hết các thao tác đều được tiến hành trong một worksheet.

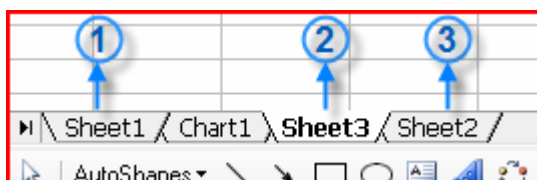


**CHÚ Ý** Worksheet chỉ là một trong nhiều dạng sheet của một workbook. Chi tiết về các loại sheet trong Excel, xem thêm mục "*Đối tượng Workbook*" trang 123.

Trong cây phân cấp đối tượng, đối tượng Worksheet nằm sau tập đối tượng Worksheets của đối tượng Workbook. Vì vậy, các đối tượng Worksheet đều được truy cập thông qua tập đối tượng Worksheets. Tương tự như đối với đối tượng Window, mỗi đối tượng Worksheet đều được truy cập thông qua tập đối tượng Worksheets bằng tên của worksheet hoặc theo thứ tự của nó.

Một cách khác để biết thứ tự của worksheet, đó là xem thứ tự xuất hiện trên thẻ chứa các sheet trong bảng tính.





Hình IV-13: Thứ tự của Worksheet

Trong minh họa trên, vị trí thứ 2 là của Chart sheet nên không được tính trong thứ tự của các worksheet.

Để tạo một worksheet mới, sử dụng phương thức Add có trong tập đối tượng Worksheets.

```
Worksheets.Add
```

Dưới đây là các phương thức và thuộc tính phổ biến trong đối tượng worksheet.

### Calculate

Phương thức này thực hiện quá trình tính toán cho toàn bộ worksheet được tham chiếu (xem thêm mục “*Calculation*” trang 121):

```
Worksheets("Sheet1").Calculate
```

### Comments

Là tập đối tượng chứa tất cả các chú thích<sup>1</sup> có trong worksheet đang được tham chiếu. Kiểu dữ liệu trả về là kiểu Comment. Đoạn mã sau sẽ hiển thị nội dung của tất cả các chú thích có trong sheet hiện hành:

```
Dim myComment As Comment
For Each myComment In Worksheets("Sheet1").Comments
    MsgBox myComment.Text
Next myComment
```

### Delete

Phương thức này sẽ xóa worksheet được tham chiếu, giống như khi chọn trình đơn **Edit⇒Delete Sheet** trong Excel. Đoạn mã sau sẽ xóa worksheet tên là “Sheet3”:

```
Worksheets("Sheet3").Delete
```

### Name

Thuộc tính này trả về tên của worksheet giống như được hiển thị trên thẻ chứa các sheet trong workbook.

### PrintOut và PrintPreview

Những phương thức này dùng để thực hiện in hoặc xem trước khi in một worksheet. Chi tiết về các tham số cho phương thức PrintOut, xem lại mục “*Đối tượng Workbook - PrintOut*” trang 125.

<sup>1</sup> Chú thích là một đoạn văn bản gắn thêm vào một ô nào đó. Đoạn văn bản này được hiện lên mỗi khi người dùng di chuột trên ô. Ô nào có chú thích thì sẽ có thêm biểu tượng hình tam giác màu đỏ ở góc trên bên phải của ô. Để tạo chú thích cho một ô, chọn trình đơn Insert⇒Comment trong Excel.

```
Worksheets("Sheet2").PrintOut PrintPreview:=True      ' In, có xem
trước
Worksheets("Sheet3").PrintPreview                    ' Xem trước khi
in
```

### Protect và Unprotect

Phương thức Protect sẽ bật chế độ bảo vệ cho worksheet giống như khi chọn từ trình đơn **Tools⇒Protection⇒Protect Sheet** trong Excel. Nếu cần tạo mật khẩu để yêu cầu người dùng nhập mỗi khi tắt chế độ bảo vệ, người dùng có thể nhập thêm vào tham số chuỗi ký tự chứa mật khẩu trong phương thức Protect

```
Worksheets("Sheet2").Protect                          ' Bật chế độ bảo vệ
Worksheets("Sheet2").Protect Password:="Excel"        ' Bảo vệ, có mật khẩu
```

Phương thức Unprotect sẽ tắt chế độ bảo vệ của worksheet. Đối với những worksheet được bảo vệ bằng mật khẩu, cần phải truyền thêm tham số là chuỗi ký tự chứa mật khẩu để tắt chế độ bảo vệ; nếu không truyền tham số mật khẩu, một hộp thoại sẽ được hiện lên để người sử dụng nhập vào mật khẩu.

```
Worksheets("Sheet2").Unprotect Password:="Excel"      ' Tắt chế độ bảo vệ
```

### Range

Đây là thuộc tính rất quan trọng trong lập trình trên Excel. Thuộc tính này sẽ được trình bày rõ hơn trong mục “*Đối tượng Range*” trang 131.

### Select

Phương thức này sẽ chọn worksheet tham chiếu làm worksheet hiện hành, tương tự như khi chọn worksheet trên thẻ chứa các sheet của workbook.

```
Worksheets("Sheet2").Select                          ' Chọn Sheet2 làm sheet hiện hành
```

### SetBackgroundPicture

Phương thức này sẽ chọn một ảnh làm ảnh nền cho worksheet, giống như khi chọn trình đơn **Format⇒Sheet⇒Background...** trong Excel. Tham số bắt buộc phải nhập vào là tên tệp đồ họa dùng để làm ảnh nền, bao gồm cả đường dẫn đầy đủ. Nếu muốn xoá ảnh nền, chỉ cần nhập tham số tên tệp đồ họa bằng rỗng.

```
Worksheets("Sheet1").SetBackgroundPicture "C:\MyPicture.jpg"
Worksheets("Sheet1").SetBackgroundPicture ""          ' Xoá ảnh
nền
```

### Visible

Thuộc tính này thiết lập sự hiển thị của worksheet, bằng TRUE nếu worksheet được hiển thị. Việc thay đổi giá trị của thuộc tính này cũng tương tự như khi chọn từ trình đơn **Format⇒Sheet⇒Hide/Unhide...** trong Excel.

```
Worksheets("Sheet1").Visible = False                ' Ẩn Sheet1
Worksheets("Sheet1").Visible = True                 ' Hiển thị lại Sheet1
```

### Name – Đặt tên cho một vùng dữ liệu trong Worksheet

Sử dụng thuộc tính Name để đặt tên cho vùng dữ liệu cần thao tác theo cách sau:

```
Dim a As Worksheet
Set a = Worksheets("Sheet1")
a.Names.Add "ABC", "=$A$1:$D$5"
```



**CHÚ Ý** Nếu tên được đặt đã có thì vùng dữ liệu cũ sẽ được định nghĩa lại theo phạm vi mới. Nếu vùng dữ liệu không có dấu \$ thì nó sẽ tự động tịnh tiến theo vị trí của ô hiện hành.

### 5.2.5. Đối tượng Range

Đối tượng Range tham chiếu đến một ô hoặc một vùng dữ liệu trên bảng tính. Đây là đối tượng phổ biến nhất trong Excel, bởi hầu hết các tương tác với Excel đều được thực hiện dựa trên các ô và vùng dữ liệu. Với đối tượng Range, người lập trình không chỉ tác động lên một ô riêng lẻ mà còn có thể tác động lên nhiều ô cùng một lúc.

#### Tham chiếu đến đối tượng Range

Việc tham chiếu đến đối tượng Range được thực hiện dựa trên địa chỉ của các ô và được thực hiện theo nhiều phương thức khác nhau. Để làm rõ hơn cách thức tham chiếu, các ví dụ sau sẽ thực hiện gán giá trị cho vùng dữ liệu được tham chiếu.

Để tham chiếu đến một ô nào đó, chỉ cần nhập địa chỉ của ô. Địa chỉ của ô có thể là kiểu địa chỉ tương đối, hoặc tuyệt đối. Ví dụ sau sẽ tham chiếu đến ô B2:

```
ActiveSheet.Range("B2").Value = 9
`hoặc có thể gán trực tiếp như sau:
ActiveSheet.Range("B2") = 9
```

	A	B	C
1			
2		9	
3			
4			

Trong trường hợp nếu người dùng có một vùng dữ liệu được đặt tên, người lập trình có thể tham chiếu đến vùng dữ liệu đó thông qua tên của vùng dữ liệu. Giả sử trong Sheet1 có một vùng dữ liệu từ ô A2 đến ô B3 được đặt tên là Input, thì cách tham chiếu như sau:

```
Worksheets("Sheet1").Range("SoLieu") = 9
```

Tên vùng dữ liệu →

	SoLieu			fx 9
	A	B	C	
1				
2	9	9		
3	9	9		
4				

Trường hợp nếu muốn tham chiếu đến một vùng dữ liệu, người lập trình có thể dựa trên địa chỉ của hai ô, ô ở góc trên bên trái và ô ở góc dưới bên phải. Ví dụ sau sẽ tham chiếu đến vùng dữ liệu từ ô B2 đến ô C3 theo nhiều cách khác nhau:

```
Worksheets("Sheet1").Range("B2:C3") = 9           `Cách thứ nhất
Worksheets("Sheet1").Range("B2.C3") = 9           `Cách thứ hai
```

## GIÁO TRÌNH TỰ ĐỘNG HOÁ THIẾT KẾ CẦU ĐƯỜNG

```
Worksheets("Sheet1").Range("B2", "C3") = 9 'Cách thứ ba
```

Hoặc thay vì sử dụng đối tượng Range, có thể dùng trực tiếp dấu ngoặc vuông ([ ]). Điều này tương đương với khi tham chiếu sử dụng đối tượng Range:

```
Worksheets("Sheet1").[B2:C3] = 9 'Cách thứ nhất  
Worksheets("Sheet1").[B2.C3] = 9 'Cách thứ hai
```

	A	B	C	D
1				
2		9	9	
3		9	9	
4				

Nếu muốn tham chiếu đến một vùng dữ liệu là giao của các vùng dữ liệu, sử dụng dấu cách giữa các vùng dữ liệu. Ví dụ sau sẽ tham chiếu đến vùng dữ liệu là giao của hai vùng dữ liệu là A1:C3 và B2:D4, vùng được tham chiếu thực sự là vùng B2:C3

```
Worksheets("Sheet1").Range("A1:C3 B2:D4") = 9
```

	A	B	C	D	E
1					
2		9	9		
3		9	9		
4					
5					

Vùng được tham chiếu

Nếu muốn tham chiếu đến một vùng dữ liệu là hợp của các vùng dữ liệu khác nhau, sử dụng dấu phẩy ngăn cách giữa các vùng dữ liệu. Ví dụ sau sẽ tham chiếu đến vùng dữ liệu là hợp của các vùng dữ liệu A1:B2, ô D3 và vùng A4:C4

```
Worksheets("Sheet1").Range("A1:B2, D3, A4.C4") = 9
```

	A	B	C	D	E
1	9	9			
2	9	9			
3				9	
4	9	9	9	9	
5					

Dưới đây là các phương thức và thuộc tính của đối tượng Range:

### Activate

Phương thức này dùng để chuyển một ô thành ô hiện hành. Nếu vùng dữ liệu là nhiều hơn một ô thì chỉ có một ô được chọn làm hiện hành, là ô ở góc trên bên trái. Cần lưu ý là phương thức này chỉ được gọi thành công nếu vùng dữ liệu đó nằm trên worksheet hiện hành. Vì vậy, muốn kích hoạt một vùng dữ liệu trên một worksheet nào đó, cần phải chuyển worksheet đó thành worksheet hiện hành.

```
Worksheets("Sheet1").Activate  
Range("A3:B5").Activate
```

### AddComment và ClearComments

Phương thức AddComment cho phép thêm chú thích vào vùng được tham chiếu. Vùng dữ liệu này chỉ được phép là một ô, và ô đó phải chưa có chú thích, nếu không sẽ làm phát sinh lỗi. Còn phương thức ClearComments cho phép xoá tất cả các chú thích của các vùng dữ liệu. Khác với phương thức AddComment, phương thức này có thể là một vùng bất kỳ.

```
Range("A1:C3").ClearComments           `Xoá chú thích vùng A1:C3
Range("B2").AddComment "Chu thích moi"  `Thêm chú thích ô B2
```

### Address

Thuộc tính này trả về địa chỉ của vùng dữ liệu được tham chiếu. Ví dụ sau sẽ hiển thị một vùng dữ liệu đã được đặt tên là SoLieu trong Sheet1:

```
MsgBox Worksheets("Sheet1").Range("SoLieu").Address
```

### BorderAround

Phương thức này thực hiện vẽ đường biên xung quanh vùng dữ liệu được tham chiếu. Người lập trình có thể thiết lập kiểu đường, bề dày nét vẽ và màu của đường.

```
Worksheets("Sheet1").Range("A1:D4").BorderAround _
LineStyle:=xlDashDot, ColorIndex:=3, Weight:=xlThick
```

### Calculate

Phương thức này thực hiện tính toán cho vùng dữ liệu được tham chiếu, áp dụng trong trường hợp chế độ tính trong Excel được thiết lập thành tính toán thủ công (Manual).

### Cells

Cells là tập đối tượng tham chiếu đến tất cả các ô nằm trong vùng dữ liệu được tham chiếu. Chi tiết xem thêm mục “Tập đối tượng Cells” trang 135.

### Clear, ClearContents và ClearFormats

Phương thức Clear xoá tất cả những gì có trong vùng dữ liệu được tham chiếu: nội dung, định dạng, chú thích...

Phương thức ClearContents chỉ xoá nội dung được lưu trữ trong vùng dữ liệu. Còn phương thức ClearFormats chỉ xoá định dạng của các ô trong vùng dữ liệu. Sau khi xoá định dạng, các ô sẽ có định dạng mặc định trong Excel.

```
Worksheets("Sheet2").Range("A1:C3").Clear           `Xoá tất cả
Worksheets("Sheet2").Range("A1:C3").ClearContents  `Xoá nội dung
Worksheets("Sheet2").Range("A1:C3").ClearFormats   `Xoá định dạng
```

### Column và Row

Hai phương thức này trả về số thứ tự của cột và hàng của ô đầu tiên của vùng dữ liệu được tham chiếu.

```
MsgBox Worksheets("Sheet1").Range("B3:D12").Column  `Hiển thị giá trị
2
MsgBox Worksheets("Sheet1").Range("B3:D12").Row     `Hiển thị giá trị
3
```

## Columns và Rows

Thuộc tính Columns và Rows thực chất là tập đối tượng kiểu Range chứa các cột và các hàng nằm trong phạm vi vùng dữ liệu được tham chiếu. Ví dụ sau sử dụng vòng lặp For Each ... Next để đổi màu và điền số thứ tự cột vào các cột trong vùng dữ liệu được tham chiếu.

```
Public Sub VD_Columns()
    Dim myColumns As Range
    For Each myColumns In Range("B3:C4,E2:F6").Columns
        myColumns.Interior.Color = RGB(0, 255, 0) 'Đổi màu
        myColumns.Value = myColumns.Column 'Điền số thứ tự
    Next myColumns
End Sub
```

	A	B	C	D	E	F	G
1							
2					5	6	
3		2	3		5	6	
4		2	3		5	6	
5					5	6	
6							
7							
8							

Vùng dữ liệu được tham chiếu



**GỢI Ý** Có thể sử dụng tập đối tượng Columns và Rows để truy cập đến cả một hàng hay một cột nào đó trong worksheet. Ví dụ sau sẽ điền giá trị 9 vào tất cả các ô trong cột C và các ô trong hàng 3:  
`Worksheets("Sheet1").Columns("C") = 9`  
`Worksheets("Sheet1").Rows("3") = 9`

## ColumnWidth và RowHeight

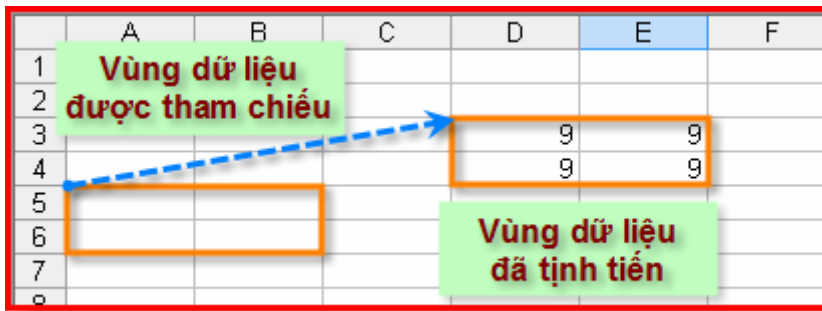
Thuộc tính này dùng để thiết lập chiều rộng của cột và chiều cao của hàng của vùng dữ liệu được tham chiếu.

```
Worksheets("Sheet2").Range("B2:C4").ColumnWidth = 15
Worksheets("Sheet2").Range("B2:C4").RowHeight = 15
```

## Offset

Hàm Offset tịnh tiến vùng dữ liệu theo số hàng và số cột được xác định trong các thông số đầu vào của hàm Offset. Giá trị trả về của hàm này chính là vùng dữ liệu sau khi đã được tịnh tiến. Cấu trúc của hàm Offset là: Offset(số\_hàng, số\_cột). Số\_hàng nếu là số dương là tịnh tiến xuống dưới, số\_cột nếu là số dương là tịnh tiến sang phải. Ví dụ sau sẽ tịnh tiến vùng dữ liệu lên trên 2 hàng và sang phải 3 cột:

```
Worksheets("Sheet1").Range("A4:B5").Offset(-2, 3).Value = 9
```



## Replace

Phương thức này dùng để thay thế một chuỗi ký tự bằng một chuỗi ký tự khác. Ví dụ sau sẽ thay thế từ SIN bằng COS:

```
Worksheets("Sheet2").Range("A1.C5").Replace "SIN", "COS"
```

Phương thức này cũng có nhiều tham số khác nữa để thiết lập chế độ tìm kiếm vào thay thế như trật tự tìm kiếm, phân biệt chữ hoa chữ thường,... Chi tiết xem trong hướng dẫn đi kèm của Excel.

## Select

Phương thức này sẽ lựa chọn vùng dữ liệu được tham chiếu, giống như khi sử dụng chuột để lựa chọn một vùng dữ liệu trong worksheet. Cũng giống như phương thức Activate, vùng dữ liệu được tham chiếu phải nằm trong worksheet hiện hành, nếu không sẽ làm phát sinh lỗi khi thực thi chương trình. Ví dụ sau sẽ chọn vùng dữ liệu B2:C3 trong worksheet hiện hành:

```
Range("B2:C3").Select
```

## Value

Thuộc tính này chứa giá trị của vùng dữ liệu. Cần phải lưu ý rằng khi đọc giá trị của vùng dữ liệu thì vùng dữ liệu đó bắt buộc phải là một ô đơn nhất, còn khi gán giá trị thì vùng dữ liệu có thể là một ô hoặc một vùng dữ liệu gồm nhiều ô và trong trường hợp đó tất cả các ô đều có cùng một giá trị.

```
MsgBox Range("A1").Value           \Đọc và hiển thị giá trị ô A1
Range("B2:C3").Value = 9           \Gán giá trị cho vùng dữ liệu B2:C3
```



**GỢI Ý** Trong khi làm việc với đối tượng Range, đối tượng tham chiếu đến một vùng dữ liệu, cần lưu ý những điểm sau:

- ◆ Việc thao tác với Excel bằng mã lệnh không cần phải thực hiện lựa chọn vùng dữ liệu, vì thế nên hạn chế sử dụng các phương thức như Activate hoặc Select.
- ◆ Trong trường hợp bắt buộc phải sử dụng các phương thức này, cần phải kích hoạt worksheet có chứa vùng dữ liệu làm worksheet hiện hành bằng phương thức Activate của worksheet đó.
- ◆ Nên sử dụng các vùng dữ liệu được đặt tên, chẳng hạn như nên sử dụng Range("KetQua") thay vì sử dụng Range("D45"). Vì khi sử dụng Range("D45"), nếu người dùng chèn thêm một hàng ở phía trên hàng 45 thì địa chỉ của ô cần tham chiếu sẽ thay đổi, và cần phải thay đổi mã lệnh thành Range("D46"). Nhưng nếu sử dụng vùng dữ liệu có đặt tên thì không cần phải thay đổi mã lệnh.
- ◆ Excel cho phép lựa chọn các vùng dữ liệu rời rạc bất kỳ. Trong khi sử dụng Excel, có thể thực hiện bằng cách giữ phím CTRL khi chọn vùng dữ liệu.

### 5.2.6. Tập đối tượng Cells

## GIÁO TRÌNH TỰ ĐỘNG HOÁ THIẾT KẾ CẦU ĐƯỜNG

Tập đối tượng Cells là tập đối tượng chứa tất cả các ô nằm trong vùng được tham chiếu. Tập đối tượng Cells là một thuộc tính của đối tượng worksheet và cũng là một thuộc tính của đối tượng Range. Khi truy cập thông qua đối tượng worksheet, tập đối tượng Cells tham chiếu đến tất cả các ô của worksheet đó. Khi truy cập thông qua đối tượng Range, tập đối tượng Cells chỉ tham chiếu đến các ô nằm trong vùng dữ liệu. Thực chất, mỗi thành phần cấu thành nên tập đối tượng Cells là một ô, có kiểu dữ liệu là Range nên tất cả các phương thức và thuộc tính của đối tượng Range đều có trong tập đối tượng Cells. Xem thêm mục “Đối tượng Range” trang 131 để biết chi tiết về đối tượng Range.

Để tham chiếu đến một ô nào đó thông qua tập đối tượng Cells, có thể sử dụng cấu trúc sau:

- ◆ `object.Cells(chi_số_hàng, chi_số_cột)`
- ◆ `object.Cells(chi_số_ô)`
- ◆ `object.Cells`

Object là đối tượng có chứa thuộc tính Cells, có thể là đối tượng kiểu Worksheet hoặc kiểu Range. Các tham số `chi_số_hàng` và `chi_số_cột` là chỉ số tương đối trong phạm vi của vùng dữ liệu được tham chiếu. `chi_số_ô` là số thứ tự của ô trong tập đối tượng Cells, số thứ tự được đánh số theo từng hàng, từ trái sang phải và từ trên xuống dưới.

Xét đoạn mã sau:

```
Worksheets("Sheet1").Range("B2:E4").Cells(2, 3).Value = 9
```

	A	B	C	D	E	F	G
1							
2							
3				9			
4							
5							
6							
7							

Đoạn mã trên sử dụng cách thứ nhất để gán giá trị 9 cho một ô nằm trong vùng B2:E4. Object ở đây chính là đối tượng kiểu Range, vì vậy tập đối tượng Cells là tập đối tượng chứa các ô trong vùng B2:E4. Chỉ số hàng và cột sẽ được tính tương đối so với ô đầu tiên của vùng dữ liệu, là ô B2. Vì vậy, `Cells(1,1)` là ô đầu tiên của vùng dữ liệu, còn `Cells(2,3)` tương ứng với ô D3.

Xét đoạn mã thứ 2:

```
Worksheets("Sheet1").Cells(257).Value = 9
```

	A	B	C	D
1				
2	9			
3				
4				

Đoạn mã trên sử dụng cách thức 2 để tham chiếu đến một ô trong worksheet. Object ở đây chính là đối tượng Worksheet, vì vậy tập đối tượng Cells là tập đối tượng chứa tất cả các ô có trong worksheet. Ô đầu tiên – ô A1 – sẽ có thứ tự là 1, các ô còn lại được đánh số từ trái sang phải và sau đó từ trên xuống dưới. Một worksheet là một vùng dữ liệu có 65536 hàng và 256 cột nên ô thứ 256 là ô cuối cùng của hàng thứ nhất, ô IV1; còn ô thứ 257 sẽ là ô đầu tiên của hàng thứ 2, ô A2.



Xét đoạn mã thứ 3:

```
Worksheets("Sheet1").Cells.Clear
```

Đoạn mã trên sử dụng cách thứ 3 để tham chiếu đến các ô. Theo đó, tất cả các ô đều được tham chiếu và sẽ đều được xử lý giống nhau.

Ngoài ra, người lập trình còn có thể tham chiếu đến từng ô trong tập đối tượng Cells bằng cách thực hiện câu lệnh lặp For Each... Next. Ví dụ sau sẽ thực hiện tính tổng tất cả các ô nằm trong vùng dữ liệu được tham chiếu:

```
Sub VD_Cells()
    Dim myCell As Range
    Dim Tong As Double
    Tong = 0
    For Each myCell In Worksheets("Sheet1").Range("A2.C4").Cells
        Tong = Tong + myCell.Value           '← Tính tổng
    Next myCell
    MsgBox Tong                               '← Hiển thị kết quả
End Sub
```

## 6. Sự kiện của các đối tượng trong Excel

Khi người dùng thực hiện một thao tác nào đó trong chương trình, Excel sẽ làm sinh một sự kiện tương ứng với các thao tác đó, chẳng hạn như các sự kiện khi mở hoặc lưu workbook. Nhờ có các sự kiện mà người lập trình có thể viết mã lệnh để thực hiện một số thao tác mỗi khi sự kiện đó xảy ra (còn gọi là bộ xử lý sự kiện – event handler). Những hộp thông báo như “Would you like to save changes?” khi ta đóng bảng tính mà chưa lưu dữ liệu là minh họa rõ nhất việc sử dụng các sự kiện trong Excel.

Thực chất, mỗi bộ xử lý sự kiện là một chương trình con dạng thủ tục. Khi sự kiện xảy ra, chương trình con tương ứng sẽ được tự động thực thi. Excel có khả năng giám sát nhiều loại sự kiện khác nhau. Các sự kiện có thể được phân loại như sau:

- ◆ Sự kiện của Workbook (sự kiện mức Workbook): các sự kiện xảy ra trong một workbook nào đó. Chẳng hạn như các sự kiện Open (khi mở hoặc tạo workbook), BeforeSave (trước khi lưu workbook), NewSheet (một sheet mới vừa được thêm),...
- ◆ Sự kiện của Worksheet (sự kiện mức Worksheet): các sự kiện xảy ra trong một worksheet nào đó. Ví dụ như các sự kiện Change (khi một ô trong sheet bị thay đổi), SelectionChange (người dùng chuyển sang vùng được chọn khác), Calculate (khi một worksheet được tính toán lại),...
- ◆ Sự kiện của đối tượng Chart: các sự kiện xảy ra trên một đối tượng chart nào đó. Chẳng hạn như các sự kiện Select (khi một đối tượng Chart được chọn), sự kiện SeriesChange (khi có một giá trị nào đó trong chuỗi số liệu bị thay đổi).
- ◆ Sự kiện của ứng dụng Excel (sự kiện mức ứng dụng): các sự kiện xảy ra bên trong chương trình Excel. Các sự kiện này bao gồm NewWorkbook (khi một workbook mới được tạo), WorkbookBeforeClose (trước khi đóng một workbook nào đó), SheetChange (khi một ô nào đó trong workbook bị thay đổi).
- ◆ Các sự kiện trong UserForm: là các sự kiện xảy ra trong UserForm hoặc trong một đối tượng nằm trên UserForm. Ví dụ như UserForm có sự kiện Initialize (xảy ra trước khi UserForm được hiển thị), hoặc đối tượng CommandButton trên UserForm có sự kiện Click (xảy ra khi người dùng kích chuột vào nút lệnh).

- ✦ Các sự kiện không gắn với đối tượng: nhóm sự kiện này có hai sự kiện rất hữu dụng: sự kiện OnTime và sự kiện OnKey. Những sự kiện này có cách thức hoạt động không giống như những sự kiện khác.

Có một số thao tác trong Excel có thể làm xảy ra nhiều sự kiện khác nhau. Ví dụ như khi người dùng chèn một worksheet mới vào trong workbook sẽ làm phát sinh các sự kiện ở mức ứng dụng như sau:

- ✦ Sự kiện WorkbookNewSheet: xảy ra khi tạo mới worksheet.
- ✦ Sự kiện SheetDeactivate: xảy ra khi worksheet hiện hành không còn hiện hành nữa.
- ✦ Sự kiện SheetActivate: xảy ra khi worksheet vừa mới được tạo được chuyển thành worksheet hiện hành.

### 6.1. Tạo bộ xử lý sự kiện cho một sự kiện

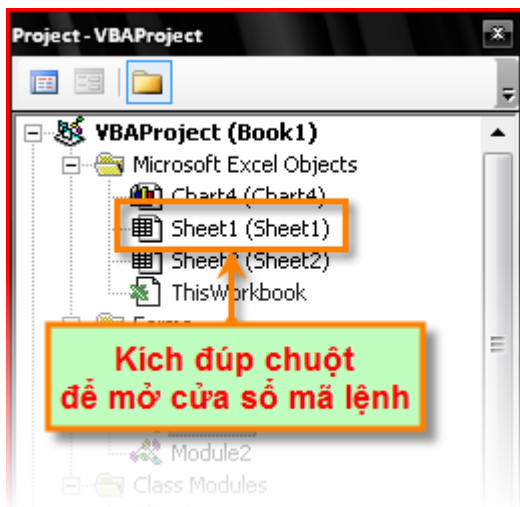
Những người mới lập trình VBA thường không biết nơi nào để tạo bộ xử lý sự kiện, hoặc bộ xử lý sự kiện được tạo ra nhưng lại không hoạt động được. Nguyên nhân là do chương trình con chứa các bộ xử lý sự kiện không được đặt đúng vị trí.

Để có thể hoạt động đúng như mong muốn, các bộ xử lý sự kiện của từng đối tượng phải được đặt trong mô-đun mã lệnh tương ứng của đối tượng đó.

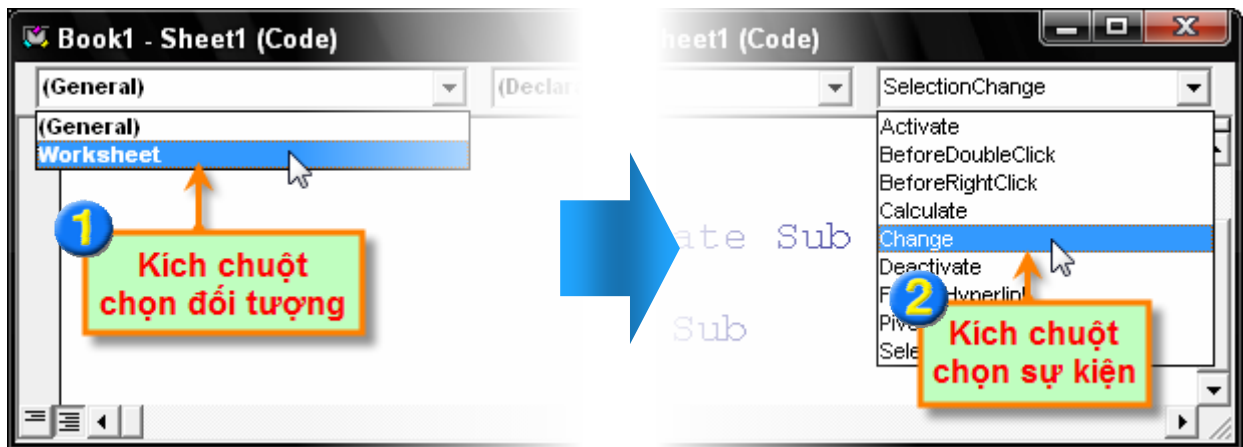
Ví dụ sau sẽ minh họa cách tạo bộ xử lý sự kiện cho sự kiện `Worksheet_Change` của Sheet 1 (là sự kiện phát sinh khi người dùng thay đổi giá trị của một ô nào đó trong Sheet 1).

#### Tạo bộ xử lý sự kiện

1. Trong cửa sổ Project của VBAIDE, kích đúp chuột lên đối tượng Sheet1 để hiển thị cửa sổ mã lệnh cho đối tượng Sheet1.



2. Trong cửa sổ mã lệnh vừa hiển thị, chọn danh sách ở góc trên bên trái và chọn mục Worksheet ⇒ chọn danh sách ở góc trên bên phải và chọn mục Change.



3. VBAIDE sẽ tự động phát sinh đoạn mã lệnh sau:

```
Private Sub Worksheet_Change(ByVal Target As Range)
End Sub
```

Chương trình con dạng thủ tục trên chính là bộ xử lý sự kiện cho sự kiện Change của đối tượng Sheet1. Người lập trình có thể viết mã lệnh để thực hiện các thao tác cần thiết mỗi khi sự kiện xảy ra. Đoạn mã sau sẽ hiển thị hộp thoại thông báo địa chỉ của ô đã bị thay đổi nội dung:

```
Private Sub Worksheet_Change(ByVal Target As Range)
    MsgBox("Ô đã bị thay đổi: " & Target.Address)
End Sub
```

Mỗi bộ xử lý sự kiện đều có các tham số riêng. Ý nghĩa và số lượng các tham số phụ thuộc vào từng loại sự kiện. Để hiểu rõ thêm về các tham số của mỗi sự kiện, tham khảo thêm trong tài liệu trợ giúp của VBA trong Excel.

**CHÚ Ý** Excel còn cho phép người dùng tắt các sự kiện trong ứng dụng, khi đó, các bộ xử lý sự kiện sẽ không được thực thi mỗi khi người dùng thực hiện các thao tác tương ứng nữa. Để tắt các sự kiện, chỉ cần gán thuộc tính `EnableEvents` của đối tượng bằng `FALSE` (`Application.EnableEvents=FALSE`). Và ngược lại, để bật lại các sự kiện, chỉ cần gán thuộc tính `EnableEvents` bằng `TRUE` (`Application.EnableEvents=TRUE`)

## 6.2. Sự kiện trong Workbook

Các sự kiện mức workbook xảy ra trong một workbook nào đó. Các bộ xử lý sự kiện của đối tượng workbook được lưu trong mô-đun mã lệnh của workbook tương ứng. Dưới đây là danh sách các sự kiện trong workbook:

Sự kiện	Thao tác làm phát sinh sự kiện
Activate	Workbook được chọn làm workbook hiện hành
AddinInstall	Workbook được cài đặt làm Add-In
AddinUninstall	Workbook bị gỡ cài đặt, không còn là Add-In nữa
BeforeClose	Ngay trước khi workbook bị đóng lại
BeforePrint	Ngay trước khi workbook được in hoặc xem trước khi in
BeforeSave	Ngay trước khi lưu workbook
Deactivate	Workbook không còn hiện hành

NewSheet	Một worksheet vừa được tạo trong workbook
Open	Mở workbook
SheetActivate	Một sheet nào đó được chọn làm sheet hiện hành
SheetBeforeDoubleClick	Người dùng kích đúp chuột trên sheet nào đó. Sự kiện này xảy ra ngay trước khi kích đúp.
SheetBeforeRightClick	Ngay trước khi người dùng kích phải chuột trên sheet
SheetCalculate	Khi trên worksheet có thực hiện tính toán nào đó
SheetChange	Khi worksheet bị thay đổi
SheetDeactivate	Khi một worksheet nào đó không còn là sheet hiện hành nữa
SheetSelectionChange	Khi người dùng thay đổi vùng lựa chọn trên worksheet
WindowActivate	Khi một cửa sổ được chọn là cửa sổ hiện hành
WindowDeactivate	Khi một cửa sổ không còn là cửa sổ hiện hành
WindowResize	Khi một cửa sổ bị thay đổi kích thước

## Sự kiện Open

Một trong những sự kiện phổ biến nhất trong Workbook chính là sự kiện Open. Sự kiện này được kích hoạt mỗi khi workbook (hoặc add-in) được mở, và sẽ kích hoạt bộ xử lý sự kiện tương ứng có tên là Workbook\_Open. Bên trong thủ tục này, người lập trình có thể thực hiện nhiều thao tác khác nhau, chẳng hạn như các thao tác phổ biến sau:

- ◆ Hiển thị một thông báo chào mừng
- ◆ Mở một workbook khác
- ◆ Thiết lập, tạo thanh trình đơn hoặc thanh công cụ
- ◆ Kích hoạt một sheet hoặc một ô nào đó
- ◆ Kiểm tra các điều kiện cần thiết khác. Chẳng hạn như kiểm tra xem add-in cần thiết cho hoạt động của workbook đã được cài đặt hay chưa...

Khuôn mẫu của bộ xử lý sự kiện Open như sau:

```
Private Sub Workbook_Open()
    'Mã lệnh sẽ được đặt ở đây
End Sub
```

Dưới đây là một ví dụ đơn giản của thủ tục Workbook\_Open. Chương trình có sử dụng hàm Weekday của VBA để xác định một ngày trong tuần. Nếu đó là ngày thứ 6, một hộp thông báo sẽ xuất hiện, nhắc nhở người dùng thực hiện sao lưu workbook hàng tuần. Nếu không phải là thứ 6, thì sẽ không có gì xảy ra cả.

```
Private Sub Workbook_Open()
    Dim strThongBao As String
    If Weekday(Now) = vbFriday Then
        strThongBao = "Hôm nay là thứ Sáu. "
        strThongBao = strThongBao & "Nhớ phải sao lưu workbook hàng tuần!"
    "
        MsgBox strThongBao, vbInformation
    End If
End Sub
```

### Sự kiện BeforeClose

Sự kiện BeforeClose xảy ra trước khi một workbook chuẩn bị đóng. Sự kiện này thường được dùng kết hợp với sự kiện Open. Lấy ví dụ như, có thể sử dụng sự kiện Open để tạo trình đơn tùy biến cho workbook, sau đó sử dụng sự kiện BeforeClose để xóa trình đơn đó trước khi workbook được đóng. Và như vậy, theo cách này, workbook luôn có một trình đơn tùy biến mà không làm ảnh hưởng đến chương trình Excel nói chung.

Khuôn mẫu của bộ xử lý sự kiện BeforeClose như sau:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    'Mã lệnh sẽ được đặt ở đây
End Sub
```

Tham số	Giải thích
Cancel	Mặc định, tham số này bằng FALSE khi xảy ra sự kiện. Nếu trong bộ xử lý sự kiện có gán giá trị cho tham số Cancel=TRUE thì Excel sẽ dừng quá trình đóng workbook lại, workbook sẽ vẫn còn được mở trong Excel.

Ví dụ sau sẽ minh họa cách thao tác với sự kiện BeforeClose. Ví dụ này sẽ kiểm tra xem khi sự kiện BeforeClose xảy ra, workbook đã được lưu hay chưa. Nếu chưa lưu sẽ hiển thị một hộp thoại yêu cầu người dùng lựa chọn các phương án: lưu – không lưu – quay trở lại workbook (không đóng workbook nữa bằng cách gán tham số Cancel = TRUE):

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Dim Msg As String
    Dim Ans As Integer
    If Not (Me.Saved) Then
        Msg = "Bạn có muốn lưu workbook: "
        Msg = Msg & Me.Name & "không ?"
        Ans = MsgBox(Msg, vbQuestion + vbYesNoCancel)
        Select Case Ans
            Case vbYes
                Me.Save
            Case vbNo
                Me.Saved = True
            Case vbCancel
                Cancel = True
        End Select
    End If
End Sub
```

Trong đoạn mã trên, khi người dùng chọn Yes thì sẽ thực hiện phương thức Save có trong đối tượng workbook. Khi người dùng chọn No thì sẽ gán thuộc tính Saved của đối tượng workbook thành TRUE, điều này sẽ làm cho Excel nghĩ là workbook đã được lưu, nhưng thực chất là không thực hiện thao tác lưu workbook. Khi người dùng chọn Cancel thì tham số Cancel sẽ được gán bằng TRUE, khi đó Excel sẽ không đóng workbook lại.

### 6.3. Sự kiện trong Worksheet

Sự kiện ở mức worksheet xảy ra bên trong một worksheet nào đó. Việc xử lý tốt các sự kiện ở mức worksheet sẽ giúp ứng dụng mở rộng hoạt động hiệu quả và chuyên nghiệp hơn. Dưới đây là một số sự kiện trong worksheet:

Sự kiện	Thao tác làm phát sinh sự kiện
---------	--------------------------------

Activate	Worksheet được chọn làm worksheet hiện hành
BeforeDoubleClick	Người dùng kích đúp chuột trên sheet. Sự kiện này xảy ra ngay trước khi kích đúp.
BeforeRightClick	Ngay trước khi người dùng kích phải chuột trên sheet
Calculate	Khi trên worksheet có thực hiện tính toán nào đó
Change	Khi một ô nào đó trong worksheet bị thay đổi
Deactivate	Worksheet không còn hiện hành
FollowHyperlink	Người dùng kích chuột vào một siêu liên kết trong worksheet
SelectionChange	Khi người dùng thay đổi vùng lựa chọn trên worksheet

Cần phải lưu ý là mã lệnh của các bộ xử lý sự kiện của worksheet phải được đặt trong mô-đun mã lệnh của worksheet tương ứng.

### Sự kiện Change

Sự kiện Change xảy ra khi có một ô nào đó trong worksheet bị thay đổi. Sự kiện này sẽ không xảy ra khi quá trình tự động tính toán của Excel làm thay đổi giá trị của ô, hoặc khi chèn một đối tượng vào trong worksheet.

Khuôn mẫu của bộ xử lý sự kiện Change như sau:

```
Private Sub Worksheet_Change(ByVal Target As Range)
    'Mã lệnh sẽ được đặt ở đây
End Sub
```

Tham số	Giải thích
Target	Tham số kiểu Range, là ô/vùng dữ liệu bị thay đổi

Khi thủ tục Worksheet\_Change được thực thi (nghĩa là khi sự kiện xảy ra), thủ tục này sẽ nhận được một đối tượng kiểu Range được truyền thông qua tham số Target. Đối tượng này có thể là một ô hoặc một vùng dữ liệu đã bị thay đổi. Ví dụ sau sẽ hiển thị một hộp thông báo thể hiện địa chỉ của ô đã bị thay đổi (địa chỉ của đối tượng Target):

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
    MsgBox "Vùng dữ liệu " & Target.Address & " đã thay đổi."
End Sub
```

Để có thể hiểu rõ hơn các loại thao tác làm phát sinh sự kiện Change của worksheet, nhập đoạn mã trên vào trong mô-đun mã lệnh của worksheet. Sau khi nhập xong đoạn mã lệnh trên, quay trở lại Excel và thực hiện thay đổi worksheet bằng nhiều cách khác nhau. Mỗi khi sự kiện Change xảy ra, một hộp thông báo sẽ được hiện lên thông báo địa chỉ của vùng dữ liệu đã bị tác động. Khi thực hiện theo cách như vậy, ta có thể tình cờ phát hiện ra nhiều điều thú vị về sự kiện này. Một số thao tác làm phát sinh sự kiện, nhưng một số thao tác khác lại không như thế:

- ✦ Thay đổi định dạng của ô không làm phát sinh sự kiện Change như mong đợi, nhưng nếu sử dụng trình đơn **Edit** ⇒ **Clear** ⇒ **Formats** thì lại làm phát sinh sự kiện này.
- ✦ Thêm, hiệu chỉnh hoặc xoá chú thích của các ô không làm phát sinh sự kiện Change.
- ✦ Nhấn phím DEL trên bàn phím sẽ làm phát sinh sự kiện Change (mặc dù ô hiện tại đang là một ô trống).

- ◆ Những ô bị thay đổi khi sử dụng các lệnh của Excel có thể có hoặc không làm phát sinh sự kiện này. Ví dụ, chọn trình đơn **Data** ⇒ **Form** và **Data** ⇒ **Sort** không làm phát sinh sự kiện. Nhưng nếu chọn trình đơn **Tools** ⇒ **Spelling** và **Edit** ⇒ **Replace** thì lại làm phát sinh sự kiện này.
- ◆ Nếu trong các chương trình con của VBA có làm thay đổi một ô nào đó thì sẽ làm phát sinh sự kiện Change.

Rõ ràng, sự kiện Change khá phức tạp và có thể có tính chất khác nhau tùy theo từng phiên bản của Excel. Tuy nhiên, sự kiện này lại rất hữu ích, đặc biệt là những ứng dụng quan trọng, đòi hỏi cần phải có sự kiểm tra, giám sát đến giá trị của từng ô.

Sự kiện Change phát sinh khi có một ô nào đó bị thay đổi, nhưng thông thường người lập trình chỉ cần quan tâm đến một vùng nào đó trong worksheet mà thôi. Ví dụ sau sẽ thực hiện giám sát một vùng dữ liệu có tên là SoLieu trong worksheet. Nếu người dùng làm thay đổi giá trị bất kỳ ô nào trong vùng này, chương trình sẽ xuất hiện thông báo cho người dùng:

```
Private Sub Worksheet_Change(ByVal Target As Range)
    Dim VRange As Range
    Set VRange = Range("SoLieu")
    If Not Intersect(Target, VRange) Is Nothing Then
        MsgBox "Ô thay đổi nằm trong vùng dữ liệu: SoLieu."
    End If
End Sub
```

Ví dụ trên sử dụng biến kiểu Range có tên là VRange, thể hiện cho vùng dữ liệu cần giám sát sự thay đổi (là vùng dữ liệu có tên là SoLieu). Thủ tục này còn sử dụng hàm Intersect của VBA, là hàm tìm giao của hai vùng dữ liệu, để kiểm tra xem vùng dữ liệu Target (được truyền qua tham số của sự kiện) có nằm trong vùng dữ liệu VRange hay không. Hàm Intersect trả về giá trị Nothing có nghĩa là hai vùng dữ liệu đó không có ô nào chung nhau. Do có sử dụng toán tử Not nên biểu thức “Not Intersect(Target, VRange) Is Nothing” sẽ trả về giá trị TRUE nếu hai vùng dữ liệu có ít nhất một ô chung nhau. Vì vậy, nếu vùng dữ liệu bị thay đổi có chung ô nào đó với vùng dữ liệu tên là SoLieu thì chương trình sẽ hiển thị hộp thông báo. Các trường hợp khác, thủ tục sẽ tự kết thúc và không có gì xảy ra cả.

#### 6.4. Sự kiện trong UserForm

Các sự kiện trên UserForm phát sinh khi có một hoạt động nào đó xảy ra – thường được phát sinh từ phía người dùng (sự kiện cũng có thể được phát sinh một cách gián tiếp từ quá trình thực hiện một phương thức nào đó). Tham khảo mục “Làm việc với UserForm và các thành phần điều khiển” trang 60 để biết thêm chi tiết.

Dưới đây là danh sách các sự kiện trong UserForm:

Sự kiện	Xảy ra khi...
Activate	UserForm được chọn là hiện hành.
Click	Người dùng kích chuột vào UserForm.
DbClick	Người dùng kích đúp chuột vào UserForm.
Deactivate	UserForm không còn là cửa sổ hiện hành.
Initialize	UserForm được tạo ra.
KeyDown	Người dùng nhấn một phím (nhưng chưa thả ra).
KeyPress	Người dùng nhấn và thả một phím.
KeyUp	Người dùng thả một phím (sau khi đã nhấn xuống).
Layout	Thay đổi kích thước hoặc vị trí của UserForm.

MouseDown	Người dùng kích chuột (nhưng chưa thả nút chuột).
MouseMove	Chuột được di chuyển trên UserForm.
MouseUp	Người dùng thả một nút chuột (sau khi đã kích chuột).
QueryClose	Trước khi UserForm bị đóng.
Resize	Thay đổi kích thước của UserForm.
Terminate	UserForm bị huỷ (UnLoad).

**6.5. Sự kiện không gắn với đối tượng**

Các sự kiện đã được đề cập đều được gắn với một đối tượng nào đó. Phần này sẽ giới thiệu một sự kiện không gắn với đối tượng nào cả rất hay dùng, đó là sự kiện và “OnKey”. Sự kiện này sẽ được truy cập thông qua đối tượng Application.

**Sự kiện OnKey**

Trong khi người dùng đang làm việc trên bảng tính, Excel luôn giám sát những gì người dùng gõ vào từ bàn phím. Vì vậy, người lập trình có thể thiết lập phím tắt (hoặc tổ hợp phím tắt) để khi người dùng nhấn phím tắt thì sẽ tự động thực thi một thủ tục mong muốn.

Để cài đặt cho sự kiện OnKey, sử dụng phương thức OnKey có trong đối tượng Application. Cú pháp của phương thức này như sau:

```
expression.OnKey Key, Procedure
```

Tham số	Giải thích
expression	Biểu thức trả về đối tượng kiểu Application
Key	Chuỗi ký tự đại diện cho phím hoặc tổ hợp phím được nhấn
Procedure	Tham số tùy chọn kiểu Variant, là chuỗi ký tự chứa tên của thủ tục sẽ được thực thi khi người dùng nhấn phím. Nếu giá trị của tham số này là "" (chuỗi rỗng) thì sẽ không có gì xảy ra khi người dùng nhấn phím cả. Nếu tham số Procedure được bỏ qua, thì phím tương ứng với tham số Key sẽ được thiết lập lại giá trị mặc định của Excel, tất cả các thủ tục đã được gán cho phím đó sẽ không còn hiệu lực nữa.

Cần lưu ý là, tham số Key có thể là một phím hoặc một tổ hợp phím kết hợp với các phím ALT, CTRL hoặc SHIFT,... Mỗi phím sẽ được đại diện bằng một ký tự hoặc chuỗi ký tự, chẳng hạn như “a” đại diện cho phím a, hay “{ENTER}” đại diện cho phím ENTER.

Để xác định các phím không hiển thị trên màn hình khi người dùng nhấn phím tương ứng (chẳng hạn như phím TAB hoặc phím ENTER) cần phải sử dụng các giá trị đã được định nghĩa sẵn. Dưới đây là danh sách các mã phím đặc biệt đó. Mỗi mã phím tương ứng với một phím trên bàn phím.

Phím	Mã phím
BACKSPACE	{BACKSPACE} hoặc {BS}
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
CLEAR	{CLEAR}
DELETE hoặc DEL	{DELETE} hoặc {DEL}
MŨI TÊN XUỐNG	{DOWN}
END	{END}

ENTER	~ (dấu ngã)
ENTER (phím số)	{ENTER}
ESC	{ESCAPE} hoặc {ESC}
F1 đến F15	{F1} đến {F15}
HELP	{HELP}
HOME	{HOME}

Phím	Mã phím
------	---------



INS	{INSERT}	SCROLL LOCK	{SCROLLLOCK}
MŨI TÊN TRÁI	{LEFT}	TAB	{TAB}
NUM LOCK	{NUMLOCK}	MŨI TÊN LÊN	{UP}
PAGE DOWN	{PGDN}	SHIFT	+ (dấu cộng)
PAGE UP	{PGUP}	CTRL	^ (dấu mũ)
RETURN	{RETURN}	ALT	% (phần trăm)
MŨI TÊN PHẢI	{RIGHT}		

Để sử dụng tổ hợp phím, chỉ cần gán tham số Key bằng hợp của tất cả các phím đơn. Ví dụ như nếu cần gán sự kiện cho tổ hợp phím CTRL+Phím cộng, gán tham số Key= “^+””; hoặc với tổ hợp phím SHIFT+CTRL+Mũi tên phải, gán tham số Key= “+^{RIGHT}”.

Ví dụ sau sẽ xử lý sự kiện OnKey để cài đặt lại chức năng của phím PgUp và phím PgDn. Sau khi thực thi thủ tục Setup\_ConKey, nếu người dùng nhấn phím PgDn, Excel sẽ thực thi thủ tục DgDn\_Sub, còn nếu người dùng nhấn phím PgUp, Excel sẽ thực thi thủ tục PgUp\_Sub. Và kết quả sẽ là: khi người dùng nhấn phím PgDn sẽ di chuyển con trỏ xuống hai hàng, còn khi nhấn phím PgUp sẽ di chuyển con trỏ lên hai hàng.

```
Sub Setup_OnKey()
    Application.OnKey "{PgDn}", "PgDn_Sub"
    Application.OnKey "{PgUp}", "PgUp_Sub"
End Sub

Sub PgDn_Sub()
    On Error Resume Next
    ActiveCell.Offset(2, 0).Activate
End Sub

Sub PgUp_Sub()
    On Error Resume Next
    ActiveCell.Offset(-2, 0).Activate
End Sub
```

Ví dụ này có sử dụng câu lệnh On Error Resume Next để bỏ qua các lỗi có thể phát sinh. Chẳng hạn như nếu ô hiện hành đang ở hàng đầu tiên, nếu cố gắng di chuyển lên trên sẽ làm phát sinh lỗi. Hoặc nếu sheet hiện hành không phải là worksheet mà là chartsheet thì cũng làm phát sinh lỗi vì không có ô hiện hành nào trên chartsheet cả.

Để xóa sự kiện OnKey cho một phím nào đấy, cần phải thực thi lại phương thức OnKey mà không có tham số Procedure. Thao tác này sẽ trả về chức năng mặc định cho phím đã được gán:

```
Sub Cancel_OnKey()
    Application.OnKey "{PgDn}"
    Application.OnKey "{PgUp}"
End Sub
```

Mặc dù bằng cách này, người lập trình có thể gán phím tắt cho một Macro bất kỳ, tuy nhiên tốt nhất là nên sử dụng trình quản lý Macro để gán phím tắt cho Macro (xem thêm mục “*Quản lý Macro*” trang 104)

## 7. Các thao tác cơ bản trong Excel

Phần này sẽ giới thiệu các đoạn mã thực hiện những thao tác cơ bản trong Excel.

## 7.1. Điều khiển Excel

Các thao tác liên quan đến việc điều khiển chương trình Excel đều được thực hiện thông qua đối tượng Application, đối tượng ở cấp cao nhất trong cây phân cấp đối tượng trong Excel.

### 7.1.1. Thoát khỏi Excel

Sử dụng phương thức có trong đối tượng Application để thoát khỏi Excel. Thông thường, nếu có workbook nào chưa được lưu, Excel sẽ hiện thị hộp thoại để nhắc người dùng lưu workbook. Tuy nhiên, người lập trình có thể thay đổi cách ứng xử trên bằng một số cách sau:

- ◆ Lưu tất cả các workbook trước khi thoát
- ◆ Gán thuộc tính Saved của workbook trước khi thoát
- ◆ Gán thuộc tính DisplayAlerts bằng FALSE

Ví dụ sau sẽ lưu tất cả các workbook đang mở trong Excel mà không cần hiển thị thông báo cho người dùng:

```
Sub QuitSaveAll( )
    Dim wb As Workbook
    For Each wb In Workbooks
        wb.Save
    Next
    Application.Quit
End Sub
```

Ngược lại, đoạn mã sau sẽ thoát khỏi Excel mà không lưu các workbook:

```
Sub QuitSaveNone()
    Dim wb As Workbook
    For Each wb In Workbooks
        ' Đánh dấu coi như các workbook đã được lưu
        ' nhưng thực chất, các workbook vẫn chưa được lưu
        wb.Saved = True
    Next
    Application.Quit
End Sub
```

Hoặc có thể sử dụng đoạn mã sau:

```
Sub QuitSaveNone()
    ' Tắt tất cả các thông báo, hộp thoại
    Application.DisplayAlerts = False
    Application.Quit
End Sub
```

Cách thoát khỏi Excel sử dụng thuộc tính Saved hoặc DisplayAlerts sẽ có thể làm mất tất cả những thay đổi chưa được lưu. Vì thế cũng có thể sử dụng một cách khác, đó là sử dụng phương thức SaveWorkspace để lưu trạng thái làm việc của Excel của trước lúc thoát vào tệp .xlw, và khi mở tệp này, trạng thái làm việc của Excel tại thời điểm đó sẽ được khôi phục lại. Đoạn mã sau sẽ lưu toàn bộ trạng thái làm việc vào tệp Resume.xlw:

```
Sub QuitWithResume()
    Application.SaveWorkspace "C:\Resume.xlw"
    Application.Quit
End Sub
```

### 7.1.2. Khóa tương tác người dùng

Trong một số trường hợp, để tránh người dùng thoát khỏi Excel khi đang thực hiện một số bước tính toán mất nhiều thời gian, cần phải giới hạn tương tác giữa người dùng và chương trình Excel. Đối tượng Application có một số phương thức/thuộc tính để thực hiện điều này:

- ◆ Gán thuộc tính `DisplayAlerts` bằng `FALSE` để ẩn các hộp thoại Excel khi đang thực thi mã lệnh.
- ◆ Gán thuộc tính `Interactive` bằng `FALSE` để người dùng hoàn toàn không thể tương tác được với Excel.
- ◆ Gán thuộc tính `ScreenUpdating` bằng `FALSE` để tắt quá trình cập nhật lại màn hình, làm ẩn đi những thay đổi diễn ra trong lúc thực thi mã lệnh.

Những cách trên đều cần phải thực hiện ở đầu đoạn mã lệnh và phải thực hiện lại ở cuối của đoạn mã lệnh để thiết lập lại các giá trị mặc định, nếu không sẽ làm khóa hoàn toàn chương trình Excel.

Ví dụ sau minh họa cách tạm thời khóa tương tác người dùng khi thực hiện các đoạn mã lệnh mất nhiều thời gian:

```
Sub Khoa_nguoi_dung()
    Dim cel As Range
    ' Chuyển con trỏ chuột thành biểu tượng chờ.
    Application.Cursor = xlWait
    ' Tắt tương tác người dùng và việc cập nhật màn hình.
    Application.Interactive = False
    Application.ScreenUpdating = False
    ' Đoạn mã lệnh mô phỏng việc tính toán mất nhiều thời gian.
    For Each cel In [a1:iv999]
        cel.Select
    Next
    ' Khôi phục lại trạng thái ban đầu.
    Application.Interactive = True
    Application.ScreenUpdating = True
    Application.Cursor = xlDefault
    [a1].Select
End Sub
```

Một lợi điểm nữa của việc gán thuộc tính `ScreenUpdating` bằng `FALSE` là việc đoạn mã lệnh sẽ thực thi với tốc độ nhanh hơn vì Excel không cần phải cập nhật lại màn hình khi tiến hành chọn từng ô trong vùng dữ liệu từ A1:IV999. Nhưng cần lưu ý phải khôi phục lại các giá trị mặc định trước khi kết thúc mã lệnh.



**GỢI Ý** Gán thuộc tính `ScreenUpdating` bằng `FALSE` trong khi thực thi các đoạn mã lệnh có liên quan đến việc hiển thị trên màn hình, chẳng hạn như đổi màu nền cho vùng dữ liệu... , sẽ làm tăng tốc độ thực thi của mã lệnh

### 7.1.3. Thao tác với cửa sổ

Đối tượng Application có tập đối tượng Windows cho phép mở, sắp xếp, thay đổi kích thước và đóng các cửa sổ bên trong Excel. Chẳng hạn như đoạn mã sau tạo thêm một cửa sổ mới và sau đó xếp chồng các cửa sổ bên trong workbook hiện hành:

```
Sub OpenCascadeWindows()
    ActiveWindow.NewWindow
    Application.Windows.Arrange xlArrangeStyleCascade, True
End Sub
```

## GIÁO TRÌNH TỰ ĐỘNG HOÁ THIẾT KẾ CẦU ĐƯỜNG

Người lập trình có thể đóng, thay đổi trạng thái của cửa sổ sử dụng các phương thức và thuộc tính có trong đối tượng Window. Ví dụ sau sẽ đóng cửa sổ đã được tạo ra ở ví dụ trước và khôi phục lại trạng thái của cửa sổ ban đầu trong Excel:

```
Sub CloseMaximize()  
    ActiveWindow.Close      'Đóng cửa sổ hiện hành  
    ActiveWindow.WindowState = xlMaximized  
End Sub
```

Việc đóng cửa sổ cuối cùng của Workbook tương đương với việc đóng workbook đó.



**CHÚ Ý** Trong cùng một workbook có thể có nhiều cửa sổ con, tất cả các cửa sổ con đó đều có nội dung giống nhau và đều là một thể hiện của workbook. Chi tiết, xem lại mục "Đối tượng Window" trang 126.

Để điều khiển cửa sổ chính của Excel, sử dụng các thuộc tính WindowState và DisplayFullScreen có trong đối tượng Application. Đoạn mã sau sẽ thực hiện thay đổi trạng thái cửa sổ chính của Excel, giữa các thái thái sẽ có một thông báo:

```
Sub ChangeExcelWindowState()  
    Application.WindowState = xlMaximized      'Phóng đại cửa sổ  
    MsgBox "Trang thái phóng đại"  
    Application.WindowState = xlMinimized      'Thu nhỏ cửa sổ  
    MsgBox "Trang thái thu nhỏ"  
    Application.WindowState = xlNormal          'Trạng thái thông thường  
    MsgBox "Trang thái thông thường"  
    Application.DisplayFullScreen = True        'Xem toàn màn hình  
    MsgBox "Trang thái toàn màn hình"  
    Application.DisplayFullScreen = False      'Trạng thái bình thường  
    MsgBox "Trang thái thông thường"  
End Sub
```

### 7.1.4. Khởi động Excel từ chương trình khác

Thông thường, khi sử dụng VBA trong Excel, chương trình Excel đã được khởi động sẵn và người lập trình không cần quan tâm đến các thao tác để khởi động chương trình Excel. Tuy nhiên, vẫn có những lúc cần khởi động chương trình Excel từ chương trình khác, chẳng hạn như khi muốn xuất dữ liệu tính toán sang Excel chẳng hạn. Lúc đó, cần phải lập trình để khởi động Excel, hay nói theo cách đơn giản hơn, là tạo đối tượng Application chứa ứng dụng Excel.

Để thực hiện được việc này, cần phải thực hiện các bước sau:

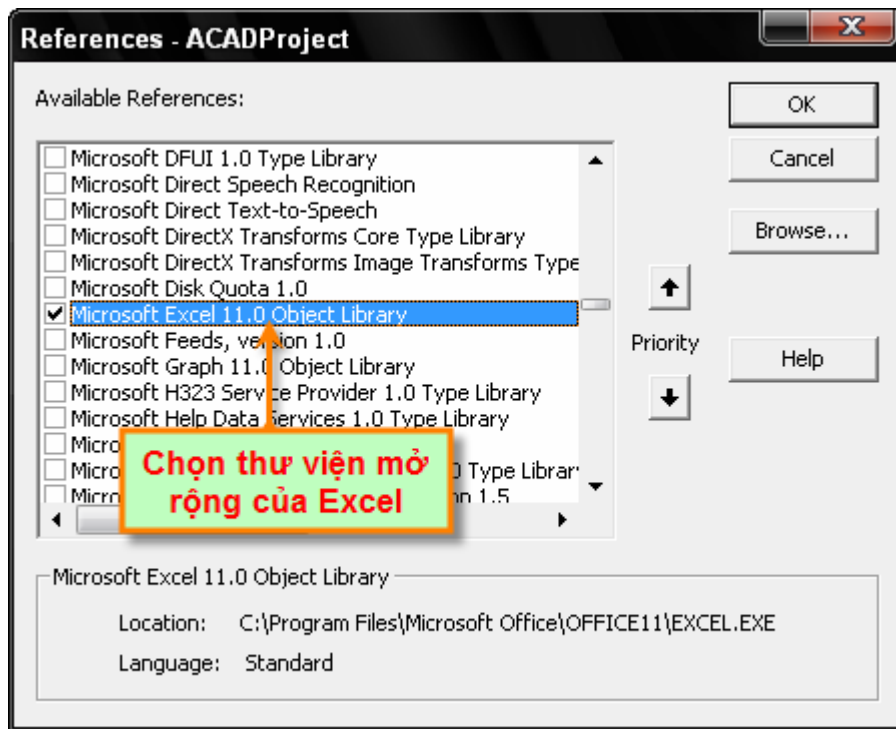
1. Tham chiếu đến với thư viện mở rộng của chương trình Excel.
2. Viết mã lệnh thực hiện việc khởi động chương trình Excel (tạo đối tượng Application của Excel).

Dưới đây sẽ trình bày cách thức khởi động chương trình Excel từ VBA trong AutoCAD.

#### Tham chiếu thư viện mở rộng của chương trình Excel

1. Khởi động chương trình AutoCAD  $\Rightarrow$  Khởi động VBAIDE trong AutoCAD bằng cách nhấn tổ hợp phím **ALT+F11**.
2. Chọn trình đơn **Tools**  $\Rightarrow$  **References...** để hiển thị hộp thoại **References** dùng để tham chiếu đến thư viện mở rộng.
3. Trong danh sách các thư viện có sẵn, chọn **Microsoft Excel 11.0 Object Library**  $\Rightarrow$  Chọn **OK**. Như vậy là dự án VBA trong AutoCAD đã có tham chiếu đến thư viện mở rộng của

Excel, nghĩa là người lập trình có thể truy cập đến mô hình đối tượng của Excel ngay từ bên trong VBA của AutoCAD



Hình IV-14: Hộp thoại References trong VBAIDE của AutoCAD.

**CHÚ Ý** Tùy từng phiên bản chương trình Excel đang được sử dụng mà tên của thư viện mở rộng Excel có thể khác nhau.

### Viết mã lệnh khởi động chương trình Excel

4. Trong VBAIDE của AutoCAD, chọn trình đơn Insert ⇒ Module để tạo mới một mô-đun chuẩn.
5. Trong cửa sổ mã lệnh của mô-đun chuẩn, nhập đoạn mã lệnh dùng để khởi động chương trình Excel như sau:

```
Sub ConnectToExcel()
    Dim App As Excel.Application
    On Error Resume Next
    Set App = GetObject(, "Excel.Application")
    ' Kiểm tra xem Excel đã được khởi động chưa
    ' Nếu chưa sẽ tiến hành tạo đối tượng Application
    If Err Then
        Err.Clear
        Set App = CreateObject("Excel.Application")
        If Err Then
            MsgBox Err.Description
            Exit Sub
        End If
    End If
    'Hiển thị cửa sổ chính của Excel
    App.Visible = True
    MsgBox "Now running " + App.Name + _
        " version " + App.Version
    '===== Kết thúc việc khởi động chương trình Excel =====
```

```
'Bắt đầu thực hiện các thao tác trong Excel  
'giống như khi thực hiện trong môi trường VBA của Excel  
Dim WBook As Workbook, WSheet As Worksheet  
Set WBook = App.Workbooks.Add  
Set WSheet = WBook.Worksheets(1)  
WSheet.Range("A1") = "Vi du ket noi voi Excel"  
WBook.SaveAs "C:\Test.xls"  
WBook.Close  
Set WBook = Nothing  
Set WSheet = Nothing  
End Sub
```

6. Thực thi Macro: **ConnectToExcel** như trên, chương trình Excel sẽ được khởi động ⇒ Tạo một Worksheet mới ⇒ Lưu thành tệp có tên `Test.xls` nằm trong thư mục gốc của ổ đĩa `C:\` ⇒ Thoát khỏi chương trình Excel.

## 7.2. Làm việc với Workbook

### 7.2.1. Tạo mới, mở, lưu và đóng workbook

#### Tạo mới workbook

Để tạo mới workbook, sử dụng phương thức `Add` có trong tập đối tượng `Workbooks`:

```
Dim wb As Workbook  
Set wb = Application.Workbooks.Add
```

#### Mở workbook

Để mở một workbook đã có, sử dụng phương thức `Open` có trong tập đối tượng `Workbooks`:

```
Dim wb As Workbook  
Set wb = Application.Workbooks.Open("C:\MyBook.xls")
```

Nếu tham số tên tệp chỉ có tên mà không bao gồm đường dẫn, Excel sẽ tìm tệp đó trong thư mục hiện hành. Nếu tệp không tồn tại thì sẽ làm phát sinh lỗi trong Excel.

#### Lưu workbook

Để lưu workbook, sử dụng phương thức `Save` và `SaveAs` có trong đối tượng `Workbook`. Thư mục mặc định để lưu trong Excel có thể được thiết lập thông qua thuộc tính `DefaultFilePath` của đối tượng `Application`, thông thường thư mục mặc định là thư mục `My Documents`. Ví dụ sau sẽ lưu workbook mới tạo vào thư mục `My Documents` với tên là `NewWorkbook.xls`:

```
ActiveWorkbook.SaveAs "NewWorkbook"
```

Phương thức `SaveAs` thích hợp khi lưu workbook lần đầu tiên, hoặc khi muốn lưu workbook thành một workbook có tên khác. Còn phương thức `Save` sẽ lưu workbook và giữ nguyên tên hiện tại của workbook.

#### Đóng workbook

Để đóng workbook, sử dụng phương thức `Close` có trong đối tượng workbook. Phương thức này không tự động lưu workbook, nhưng khi có sự thay đổi nào đó chưa lưu, Excel sẽ hiển thị hộp thoại `SaveChanges` trước khi đóng workbook. Người lập trình có thể tắt hộp thoại này bằng cách truyền thêm tham số vào cho phương thức `Close`:

```
ThisWorkbook.Close True
```

Đoạn mã sẽ lưu tất cả các thay đổi và sau đó đóng workbook hiện hành. Để đóng mà không lưu những thay đổi của workbook, sử dụng đoạn mã sau:

```
ThisWorkbook.Close False
```

Để đóng tất cả các workbook, sử dụng phương thức Close có trong tập đối tượng Workbooks. Tuy nhiên phương thức này không có tham số, vì vậy hộp thoại SaveChanges sẽ xuất hiện khi có workbook nào đó chưa được lưu.

```
Sub TestCloseAll()  
    Workbooks.Close  
End Sub
```

### 7.3. Làm việc với Worksheet

#### 7.3.1. Tạo mới, xoá và đổi tên worksheet

##### Tạo mới worksheet

Để tạo mới Worksheet, sử dụng phương thức Add có trong tập đối tượng Worksheets hoặc tập đối tượng Sheets.

```
Sub Tao_moi_worksheet()  
    Dim ws1 As Worksheet  
    Dim ws2 As Worksheet  
    'Thêm một worksheet vào trước worksheet hiện hành  
    Set ws1 = Worksheets.Add  
    'Thêm một worksheet khác vào sau sheet cuối cùng của workbook  
    Set ws2 = Sheets.Add(After:=Sheets(Sheets.Count), Type:=xlWorksheet)  
End Sub
```

Do tập đối tượng Sheets bao gồm nhiều loại sheet khác nhau nên khi thêm worksheet sử dụng phương thức Add có trong tập đối tượng Sheets, cần phải xác định rõ loại sheet sẽ được thêm vào.

##### Xoá worksheet

Để xoá worksheet, sử dụng phương thức Delete có trong đối tượng worksheet. Ví dụ sau sẽ xoá worksheet có tên là Sheet1.

```
Sub Xoa_worksheet()  
    Dim mySheet As Worksheet  
    Set mySheet = Worksheets("Sheet1")  
    Application.DisplayAlerts = False  
    mySheet.Delete  
    Application.DisplayAlerts = True  
End Sub
```

Khi sử dụng phương thức Delete để xoá worksheet, Excel sẽ hiển thị hộp thông báo để xác nhận thao tác xoá worksheet. Vì vậy, khi đã chắc chắn xoá worksheet phải tắt hết tất cả các thông báo bằng cách gán thuộc tính DisplayAlerts bằng FALSE, và cũng nên nhớ là phải trả về các giá trị mặc định trước khi kết thúc chương trình.

##### Đổi tên worksheet

Việc đổi tên worksheet được thực hiện một cách dễ dàng, chỉ cần thay đổi thuộc tính Name có trong đối tượng worksheet. Đoạn mã sau sẽ đổi tên Sheet2 thành MySheet:

```
Worksheets("Sheet2").Name = "MySheet"
```

## 7.4. Làm việc với Range và Cells

### 7.4.1. Duyệt qua từng ô trong vùng dữ liệu

Để duyệt qua từng ô trong vùng dữ liệu, sử dụng vòng lặp For Each... Next để duyệt qua từng đối tượng trong tập đối tượng Cells. Trình tự duyệt là theo số thứ tự của ô: duyệt từ trái sang phải là từ trên xuống dưới. Ví dụ sau sẽ duyệt qua từng ô trong vùng dữ liệu A1:D3, điền số vào từng ô theo thứ tự được duyệt. Thông qua ví dụ này, ta sẽ hiểu rõ hơn về trình tự duyệt các ô trong vùng dữ liệu:

```
Sub Duyet_O()
    Dim myCell As Range
    Dim i As Integer
    i = 0
    For Each myCell In Range("A1:D3").Cells
        'Các thao tác xử lý nằm ở đây
        'Ví dụ: điền số thứ tự duyệt vào từng ô
        i = i + 1
        myCell.Value = i
    Next myCell
End Sub
```

	A	B	C	D	E	F
1	1	2	3	4		
2	5	6	7	8		
3	9	10	11	12		
4						
5						

Vùng dữ liệu duyệt

### 7.4.2. Duyệt qua từng ô trong vùng dữ liệu theo hàng và cột

Quá trình duyệt theo hàng hoặc cột được thực hiện sử dụng vòng lặp For Each...Next trên các tập đối tượng Rows, Columns và Cells. Đối tượng thành phần trong các tập đối tượng này đều có kiểu là Range. Ví dụ sau sẽ tính tổng từng cột của vùng dữ liệu được tham chiếu và điền giá trị tổng này vào ô ở ngay phía dưới từng cột.

```
1. Sub Duyet_O_Theo_Cot()
2. Dim myCell As Range
3. Dim myColumn As Range
4. Dim Tong As Double
5. For Each myColumn In Range("A1:D3").Columns
6. Tong = 0
7. For Each myCell In myColumn.Cells
8. Tong = Tong + Val(myCell.Value)
9. Next myCell
10. myColumn.Cells(myColumn.Rows.Count + 1, 1) = Tong
11. Next myColumn
12. End Sub
```

Dòng thứ 5 là vòng lặp cho phép duyệt qua từng cột trong vùng dữ liệu A1:D3. Mỗi cột lại là một vùng dữ liệu, vì thế lại tiếp tục duyệt qua từng ô trong vùng dữ liệu đó, điều này được thực hiện ở dòng thứ 7. Dòng thứ 10 dùng để gán giá trị tổng tính được của mỗi ô vào ô dưới cùng của cột.



	A	B	C	D	E	F	G
1	1	3	5	7			
2	2	4	3	4			
3	1	2	3	3			
4	4	9	11	14			
5							
6							

Vùng dữ liệu được tham chiếu (A1:D4)  
Tổng các cột (D4)

### 7.4.3. Vùng có chứa dữ liệu – Thuộc tính UsedRange

UsedRange là một thuộc tính rất hữu dụng của đối tượng Worksheet. Thuộc tính này trả về vùng dữ liệu là hình chữ nhật bao của tất cả các ô có chứa dữ liệu. Góc trên bên trái của hình chữ nhật là ô đầu tiên có chứa dữ liệu, còn góc dưới bên phải của hình chữ nhật là ô cuối cùng có chứa dữ liệu. Các ô có chứa dữ liệu được hiểu là những ô có chứa thông tin như: giá trị, định dạng và chú thích. Hình sau minh họa rõ hơn về thuộc tính UsedRange.

	A	B	C	D	E	F	G	H
2								
3								
4				10				
5								
6								
7								
8								
9								
10								

Ô chứa giá trị (D4)  
Ô chứa định dạng (C5)  
Ô chứa chú thích (D8)  
UsedRange (C4:D8)

Mặc dù trong vùng dữ liệu trả về của thuộc tính UsedRange có chứa cả những ô không có dữ liệu, nhưng như vậy đã là hiệu quả và tiết kiệm thời gian hơn là so với việc duyệt qua tất cả các ô trong worksheet. Ví dụ sau sẽ duyệt qua tất cả các ô có chứa dữ liệu và chọn những ô có giá trị âm trên worksheet hiện hành:

```
Sub Su_dung_UsedRange( )
    Dim cel As Range, str As String
    For Each cel In ActiveSheet.UsedRange
        If cel.Value < 0 Then str = str & cel.Address & ","
    Next
    If str <> "" Then
        str = Left(str, Len(str) - 1)
        ActiveSheet.Range(str).Select
    End If
End Sub
```

Ví dụ trên lấy về địa chỉ của tất cả các ô có giá trị âm và sử dụng dấu “,” ngăn cách giữa địa chỉ của các ô để lấy hợp của tất cả các ô (xem thêm mục “Tham chiếu đến đối tượng Range” trang 131). Sau khi kết thúc vòng lặp, chuỗi str sẽ có kiểu là “\$A\$1,\$D\$5,” nên dòng lệnh If cuối cùng sẽ cắt ký tự cuối cùng của chuỗi str để chuyển về dạng thức địa chỉ đúng “\$A\$1,\$D\$5”. Câu lệnh Len(str) trả về chiều dài của chuỗi ký tự str. Còn câu lệnh Left(str,n) trả về n ký tự nằm ở bên trái của chuỗi ký tự str.

### 7.5. Làm việc với biểu đồ

Tính năng biểu đồ trong Excel khá ấn tượng. Một biểu đồ có thể thể hiện nhiều kiểu dữ liệu khác nhau trong Excel. Excel hỗ trợ hơn 100 loại biểu đồ khác nhau và người dùng có thể điều khiển hầu như tất cả các thành phần trong biểu đồ bởi lẽ, mỗi thành phần trong biểu đồ chính là một đối tượng với các phương thức và thuộc tính khác nhau. Vì vậy, việc lập trình với biểu đồ

là không dễ, nhưng một khi đã hiểu rõ cây phân cấp đối tượng thì kết quả sẽ ấn tượng hơn rất nhiều.

Tùy theo vị trí mà biểu đồ trong Excel được phân thành 2 loại sau:

- ✦ Biểu đồ nhúng – ChartObject: là dạng biểu đồ nằm bên trong một worksheet. Trong một worksheet có thể chứa nhiều biểu đồ nhúng khác nhau và các biểu đồ này có thể được truy xuất thông qua tập đối tượng ChartObjects có trong đối tượng worksheet.
- ✦ Biểu đồ độc lập – ChartSheet: là dạng biểu đồ nằm trong một sheet riêng biệt, gọi là chartsheet. Mỗi một chartsheet chỉ có thể chứa một biểu đồ dạng này mà thôi. Biểu đồ dạng này có thể được truy xuất thông qua tập đối tượng Charts có trong đối tượng workbook.

Biểu đồ, dù là dạng nhúng hay độc lập, đều có cùng một kiểu dữ liệu là Chart. Hơn nữa, trong hầu hết các bảng tính, các biểu đồ thường được nhúng trong worksheet để tiện cho việc trình bày. Chính vì vậy, nội dung trong giáo trình này chỉ tập trung thao tác đối với biểu đồ nhúng.

### 7.5.1. Tạo mới biểu đồ

Cách nhanh nhất để tạo biểu đồ bằng mã lệnh là sử dụng phương thức ChartWizard của đối tượng Chart. Với phương thức này, người lập trình có thể tạo được biểu đồ chỉ trong 2 bước:

1. Tạo mới đối tượng Chart, sử dụng phương thức Add.
2. Gọi phương thức ChartWizard của đối tượng Chart vừa được tạo.

Phương thức ChartWizard có rất nhiều tham số khác nhau, tất cả đều là tham số tùy chọn.

**ChartWizard(Source, Gallery, Format, PlotBy, CategoryLabels, SeriesLabels, HasLegend, Title, CategoryTitle, ValueTitle, ExtraTitle)**

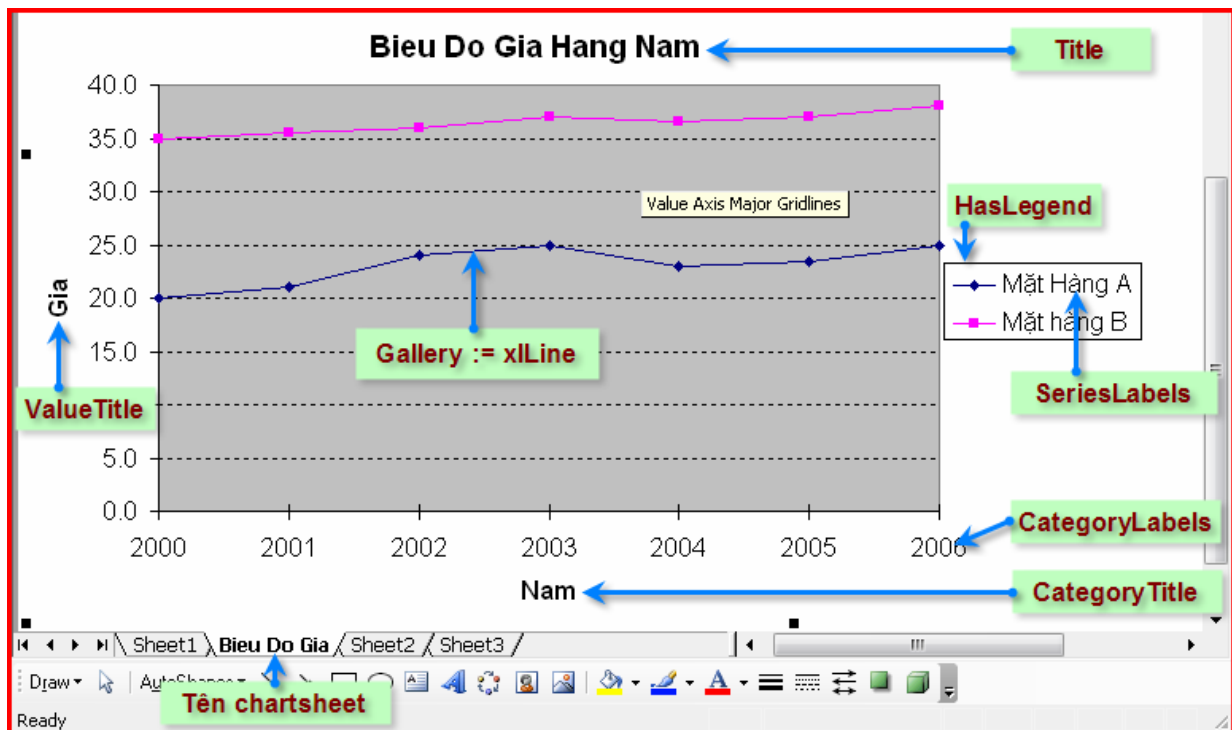
Tham số	Giải thích
Source	Vùng dữ liệu chứa số liệu đầu vào cho biểu đồ
Gallery	Dạng biểu đồ, có thể là một trong những giá trị sau: xlArea, xlBar, xlColumn, xlLine, xlPie, xlRadar, xlXYScatter, xlCombination, xl3DArea, xl3DBar, xl3DColumn, xl3DLine, xl3DPie, xl3DSurface, xlDoughnut, xlDefaultAutoFormat.
Format	Định dạng tự động. Giá trị từ 1÷10 tùy thuộc vào loại biểu đồ. Nếu bỏ qua tham số này, Excel sẽ tự chọn giá trị mặc định dựa trên dạng biểu đồ và số liệu đầu vào.
PlotBy	Xác định xem số liệu cho từng chuỗi số liệu là theo cột hay hàng, có thể là xlRows hoặc xlColumns.
CategoryLabels	Số nguyên xác định số hàng hoặc cột bên trong vùng dữ liệu đầu vào sẽ làm CategoryLabels.
SeriesLabels	Số nguyên xác định số hàng hoặc cột bên trong vùng dữ liệu đầu vào sẽ làm SeriesLabels.
HasLegend	Bằng TRUE thì biểu đồ sẽ có thêm phần chú giải.
Title	Tiêu đề của biểu đồ
CategoryTitle	Tiêu đề của trục ngang
ValueTitle	Tiêu đề của trục đứng
ExtraTitle	Tiêu đề trục đối với biểu đồ 3D hoặc tiêu đề của trục giá trị thứ 2 của biểu đồ 2D

Ví dụ sau tạo một biểu đồ trong một chartsheet nằm sau worksheet hiện hành, sau đó sử dụng phương thức ChartWizard để tạo biểu đồ dựa trên vùng dữ liệu có tên là SoLieu. Nội dung của vùng dữ liệu đó như sau:

	Mặt Hàng A	Mặt hàng B
2000	20.0	35.0
2001	21.0	35.5
2002	24.0	36.0
2003	25.0	37.0
2004	23.0	36.5
2005	23.5	37.0
2006	25.0	38.0

```

Sub ChartWizard()
    Dim ws As Worksheet, chrt As Chart
    Set ws = ActiveSheet
    ' Tạo mới chartsheet, nằm sau worksheet hiện hành.
    Set chrt = Charts.Add(, ws)
    ' Đặt tên cho chartsheet.
    chrt.Name = "Bieu Do Gia"
    ' Tạo biểu đồ sử dụng phương thức ChartWizard.
    chrt.ChartWizard ws.[SoLieu], xlLine, , xlColumns, 1, 1, True, _
        "Bieu Do Gia Hang Nam", "Nam", "Gia"
End Sub
    
```



### 7.5.2. Thêm một chuỗi số liệu vào biểu đồ đã có

Các số liệu đã được vẽ trong biểu đồ được lưu trữ trong tập đối tượng SeriesCollection. Mỗi hàng hoặc cột dữ liệu được lưu trữ trong một đối tượng Series tương ứng và mỗi số liệu trong một đối tượng Series được lưu trữ trong đối tượng Point.

## GIÁO TRÌNH TỰ ĐỘNG HOÁ THIẾT KẾ CẦU ĐƯỜNG

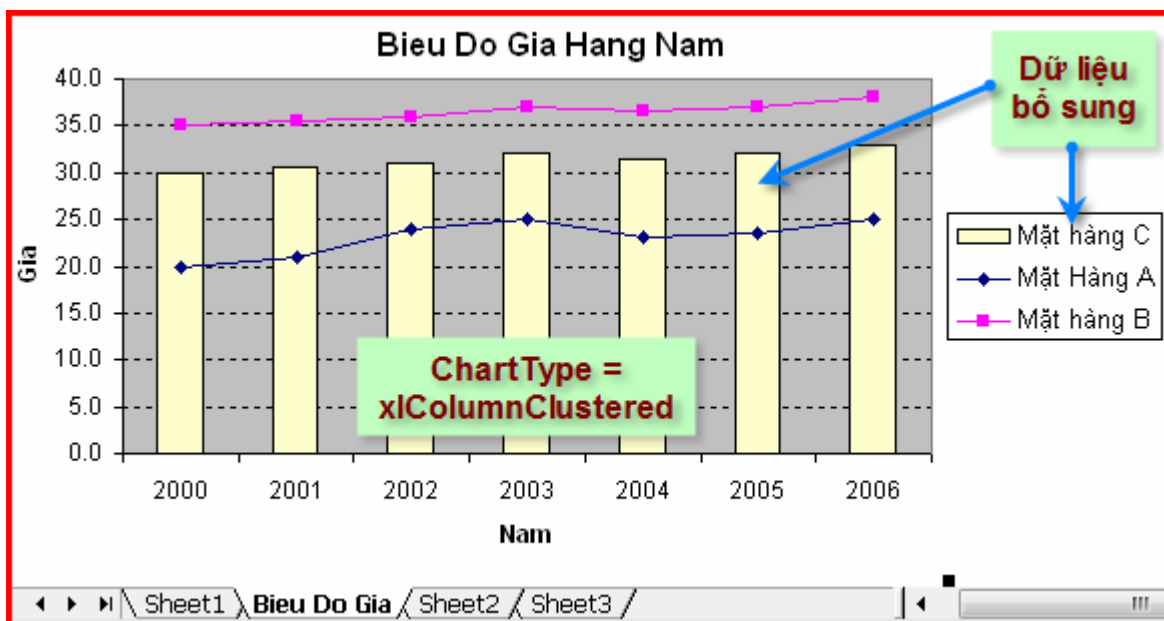
Người lập trình có thể thêm chuỗi số liệu vào một biểu đồ đã có bằng cách gọi phương thức Add của tập đối tượng SeriesCollection. Phương thức Add có những tham số sau:

Tham số	Giải thích
Source	Vùng dữ liệu chứa dữ liệu của chuỗi số liệu mới hoặc có thể là mảng số liệu
Rowcol	Xác định xem chuỗi số liệu bố trí theo dạng cột hay dạng hàng, có thể là xlRows hoặc xlColumns.
SeriesLabels	Giá trị này sẽ bị bỏ qua nếu Source là một mảng số liệu. Nếu Source là vùng dữ liệu thì giá trị này sẽ là TRUE nếu hàng hoặc cột đầu tiên của vùng dữ liệu chứa tên của chuỗi số liệu, nếu không thì gán giá trị này bằng FALSE.
CategoryLabels	Giá trị này sẽ bị bỏ qua nếu Source là một mảng số liệu. Nếu Source là vùng dữ liệu thì giá trị này sẽ là TRUE nếu hàng hoặc cột đầu tiên của vùng dữ liệu chứa giá trị làm CategoryLabels của chuỗi số liệu, nếu không thì gán giá trị này bằng FALSE.
Replace	Nếu CategoryLabels là TRUE và Replace là TRUE, thì giá trị CategoryLabels của biểu đồ hiện tại sẽ được thay mới. Nếu Replace là FALSE thì CategoryLabels của biểu đồ hiện tại sẽ được giữ nguyên. Mặc định là giá trị FALSE.

Ví dụ sau sẽ thêm một chuỗi số liệu mới vào biểu đồ đã được tạo ở ví dụ trước, và chuyển dạng biểu đồ thành dạng cột.

```

Sub AddNewSeries ()
Dim chrt As Chart, sc As SeriesCollection, sr As Series
' Lấy lại biểu đồ theo tên biểu đồ.
Set chrt = Charts("Bieu Do Gia")
' Lấy tập đối tượng SeriesCollection.
Set sc = chrt.SeriesCollection
' Thêm chuỗi số liệu mới.
sc.Add [ThemSoLieu], xlColumns, True, False, False
' Lấy chuỗi số liệu cuối trong tập đối tượng SeriesCollection
' chính là chuỗi số liệu mới bổ sung.
Set sr = sc(sc.Count)
' Đổi dạng biểu đồ cho chuỗi số liệu mới.
sr.ChartType = xlColumnClustered
End Sub
    
```



## 7.6. Sử dụng các hàm có sẵn trong Excel

Người lập trình có thể tận dụng các hàm có sẵn của Excel trong khi lập trình trên VBA thông qua đối tượng WorksheetFunction. Đối tượng này là một thuộc tính của đối tượng gốc Application.

Ví dụ sau sẽ tìm giá trị nhỏ nhất trên vùng dữ liệu A1:A10 bằng cách sử dụng hàm Min của Excel:

```
Set myRange = Worksheets("Sheet1").Range("A1:C10")
answer = Application.WorksheetFunction.Min(myRange)
MsgBox answer
```

## 8. Giao diện người dùng

Khi xây dựng chương trình, để người khác có thể dùng được, người lập trình cần phải đặc biệt chú ý đến giao diện người dùng. Giao diện người dùng được hiểu là cách thức mà người sử dụng sẽ tương tác với chương trình bằng cách nhấn nút bấm, chọn một trình đơn, nhấn phím, chọn trên thanh công cụ,...

Khi xây dựng các ứng dụng, cần phải luôn ghi nhớ rằng mục đích xây dựng chương trình là để cho người dùng cuối sử dụng. Người lập trình thường có kinh nghiệm sử dụng máy tính hơn người dùng, cho nên, với một giao diện nào đó thì đối với người lập trình là dễ sử dụng trong khi đó, đối với người sử dụng lại rất khó dùng.

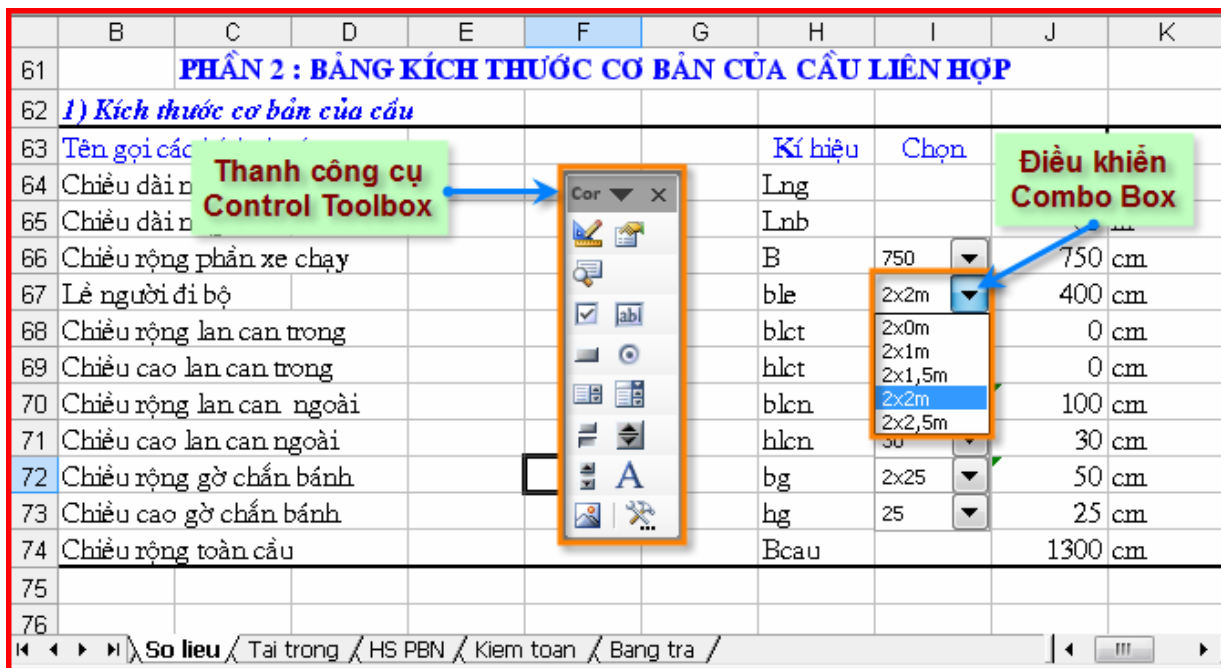
Khi một chương trình được triển khai xây dựng dựa trên VBA của Excel thì hợp lý nhất là nên hướng đến việc sử dụng những tính năng sẵn có của chính Excel làm giao diện, có như vậy ta mới tận dụng được một trong những thế mạnh của Excel, đó là giao diện thân thiện, đơn giản và hiệu quả. Với định hướng thiết kế giao diện như vậy, ta nên sử dụng hệ thống trình đơn, thanh công cụ và chính bảng tính làm giao diện chính cho ứng dụng của mình.

Như vậy trong Excel, người dùng có thể sử dụng những tính năng được cung cấp sẵn để thiết kế giao diện cho chương trình của mình và sau đây là một số phương án thiết kế giao diện nên sử dụng khi lập trình VBA trong Excel:

- ◆ Sử dụng điều khiển nhúng trực tiếp trên worksheet chẳng hạn như ListBox hoặc CommandButton;
- ◆ Sử dụng các hộp thoại thông dụng có sẵn trong Excel;
- ◆ Tạo các hộp thoại tùy biến (chính là việc sử dụng UserForm);
- ◆ Tùy biến trình đơn;
- ◆ Tùy biến thanh công cụ;
- ◆ Tùy biến phím tắt.





### 8.1. Điều khiển nhúng trong Worksheet

Điều khiển nhúng trong Worksheet, hay còn gọi là điều khiển ActiveX, là những điều khiển có thể chèn trực tiếp vào trong worksheet, liên kết trực tiếp với dữ liệu trong các worksheet mà không cần thêm một đoạn mã lệnh nào khác. Đương nhiên, nếu cần thì người lập trình có thể thêm các đoạn mã lệnh để xử lý các tình huống khác cho từng điều khiển. Thanh công cụ Toolbox sẽ giúp cho người dùng thực hiện thiết kế giao diện kiểu này.




Hình IV-15: Bảng tính sử dụng điều khiển nhúng trong worksheet.

Để hiển thị thanh công cụ Control Toolbox, chọn trình đơn **View**⇒**Toolbars**⇒**Control Toolbox**. Trên thanh công cụ này, cần chú ý đến 3 biểu tượng đầu tiên phục vụ cho quá trình thiết kế các điều khiển trong worksheet:

- ◆ Design Mode  : khi biểu tượng này được hiện sáng () , tức là các điều khiển đang ở trong chế độ thiết kế. Ở chế độ này, người lập trình có thể chọn các điều khiển, thay đổi các thuộc tính của chúng... Khi biểu tượng này ở chế độ thông thường, tức là các điều khiển đang ở trong chế độ thực thi. Ở chế độ này, các điều khiển sẽ ở trạng thái sử dụng.
- ◆ Properties  : nhấn chuột vào biểu tượng này sẽ hiển thị cửa sổ Properties, liệt kê tất cả các thuộc tính của điều khiển được chọn. Thông qua cửa sổ này, người lập trình có thể thay đổi từng thuộc tính liên quan đến điều khiển được chọn.
- ◆ View Code  : nhấn chuột vào biểu tượng này sẽ hiển thị cửa sổ mã lệnh tương ứng với điều khiển được chọn.

Trên thanh công cụ Control Toolbox còn có nhiều biểu tượng khác nữa, mỗi biểu tượng tương ứng với một điều khiển. Về cơ bản, các điều khiển này tương đương với các điều khiển đã được trình bày ở phần trước (tham khảo mục “*Các điều khiển thông dụng*” trang 69).

### 8.1.1. Điều khiển Spin Button

Spin button,  , là một nút bấm gắn với một ô nào đó trong worksheet. Để tăng giá trị trong ô đó, người dùng sẽ bấm vào mũi tên lên, còn để giảm giá trị, người dùng sẽ bấm vào mũi tên xuống.


Spin button thích hợp khi muốn hạn chế số liệu nhập vào nằm trong một giới hạn nào đó.

Các thuộc tính cơ bản của Spin button:

Thuộc tính	Mô tả
Name	Kiểu String. Tên của điều khiển
LinkedCell	Kiểu String. Địa chỉ của ô sẽ liên kết trực tiếp với điều khiển
Max	Kiểu Integer. Giá trị lớn nhất có thể đạt được

Min	Kiểu Integer. Giá trị nhỏ nhất có thể đạt được
SmallChange	Kiểu Integer. Số gia mỗi khi người dùng nhấn vào mũi tên lên hoặc xuống
Value	Kiểu Integer. Giá trị hiện thời của điều khiển, cũng là giá trị sẽ hiển thị trong ô liên kết trực tiếp với điều khiển

### 8.1.2. Điều khiển ComboBox

Combo Box, , sử dụng để người dùng lựa chọn một phần tử trong danh sách đổ xuống. Điều khiển này thường được sử dụng khi người lập trình muốn người sử dụng chỉ có thể chọn được những phần tử đã được định trước, tránh những sai sót trong quá trình nhập dữ liệu, chẳng hạn như chỉ cho phép người dùng chọn một trong các loại máy của bê tông mà chương trình hỗ trợ. Thông thường ta nên đặt Combo Box trùng lên ô mà nó liên kết.

Các thuộc tính cơ bản của Combo Box:

Thuộc tính	Mô tả
Name	Kiểu String. Tên của điều khiển
LinkedCell	Kiểu String. Địa chỉ của ô liên kết trực tiếp với Combo Box. Giá trị của Combo Box chính là giá trị của ô được liên kết.
ListFillRange	Kiểu String. Địa chỉ của vùng dữ liệu cấu thành danh sách các phần tử trong Combo Box. Mỗi một hàng của vùng dữ liệu là một phần tử trong danh sách đó.
ColumnCount	Kiểu Integer. Số cột sẽ được hiển thị trong danh sách xổ xuống của Combo Box. Mặc định ColumnCount=1.
BoundColumn	Kiểu Integer. Số thứ tự cột trong vùng dữ liệu, là cột mà giá trị của cột đó sẽ được gán cho thuộc tính Value của Combo Box khi một phần tử trong Combo Box được chọn.
ColumnHeads	Kiểu Boolean. Nếu bằng FALSE, không hiển thị phần tiêu đề của cột trong danh sách xổ xuống. Nếu bằng TRUE, hiển thị tiêu đề của cột trong danh sách xổ xuống, và hàng dữ liệu nằm ngay phía trên ListFillRange sẽ được lấy làm tiêu đề của cột.
ColumnWidths	Bề rộng của cột, tính bằng pt. Nếu có nhiều cột thì bề rộng của mỗi cột sẽ được cách nhau bằng dấu chấm phẩy (;). Ví dụ 60:80
Style	Kiểu Integer. Nếu bằng 0, người dùng có thể nhập dữ liệu trực tiếp vào Combo Box hoặc chọn từ danh sách. Nếu bằng 1, người dùng chỉ có thể nhập dữ liệu bằng cách chọn từ danh sách.
Text	Kiểu String. Là đoạn văn bản/dữ liệu được hiển thị trong Combo Box.
Value	Kiểu Variant. Là giá trị thực tế của phần tử được chọn trong Combo Box. Giá trị của thuộc tính Value này chính là giá trị của ô được liên kết với Combo Box thông qua thuộc tính LinkedCell.

### Tạo Combo Box có nhiều cột

Ví dụ sau sẽ làm rõ ý nghĩa của các thuộc tính trên thông qua việc tạo một Combo Box có nhiều cột. Combo Box sẽ hiển thị các cấp đường thiết kế, và khi lựa chọn một cấp đường, giá trị vận tốc thiết kế tương ứng sẽ được gán cho ô. Trình tự thực hiện như sau:

1. Tạo mới Combo Box trên worksheet, di chuyển đến vị trí thích hợp.
2. Chọn kiểu cho Combo Box bằng cách gán thuộc tính `Style` bằng 1, nghĩa là người dùng chỉ có thể lựa chọn từ danh sách xổ xuống.
3. Gán giá trị cho các thuộc tính `LinkedCell` bằng C11 là ô sẽ chứa giá trị vận tốc thiết kế được chọn. Gán giá trị `ListFillRange` bằng F2:G7 là vùng dữ liệu chứa bảng các giá trị

## GIÁO TRÌNH TỰ ĐỘNG HOÁ THIẾT KẾ CẦU ĐƯỜNG


cấp đường và vận tốc thiết kế. Thông thường vùng dữ liệu này sẽ được lưu trữ trong một worksheet khác hoặc một nơi mà người dùng không nhìn thấy được để tránh gây ra sự lúng túng cho người dùng.

- Để hiển thị được nhiều cột, gán giá trị thuộc tính `ColumnCount` bằng 2. Do giá trị cần liên kết là giá trị vận tốc thiết kế, tức là giá trị nằm ở cột thứ 2 của vùng dữ liệu, do vậy cần phải gán thuộc tính `BoundColumn` bằng 2.
- Để hiển thị tiêu đề cho danh sách xổ xuống, gán giá trị `ColumnHeads` bằng `TRUE`.

	A	B	C	D	E	F	G
1	Chọn cấp đường thiết kế					Cấp thiết kế	Vận tốc thiết kế
2						I	120
3	IV					II	100
4	Cấp thiết kế	Vận tốc thiết kế				III	80
5	I	120				IV	60
6	II	100				V	40
7	III	80				VI	30
8	IV	60					
9	V	40					
10	VI	30					
11	Vận tốc TK được chọn		60				
12							
13							
14							
15							

Hình IV-16: Combo Box có nhiều cột.

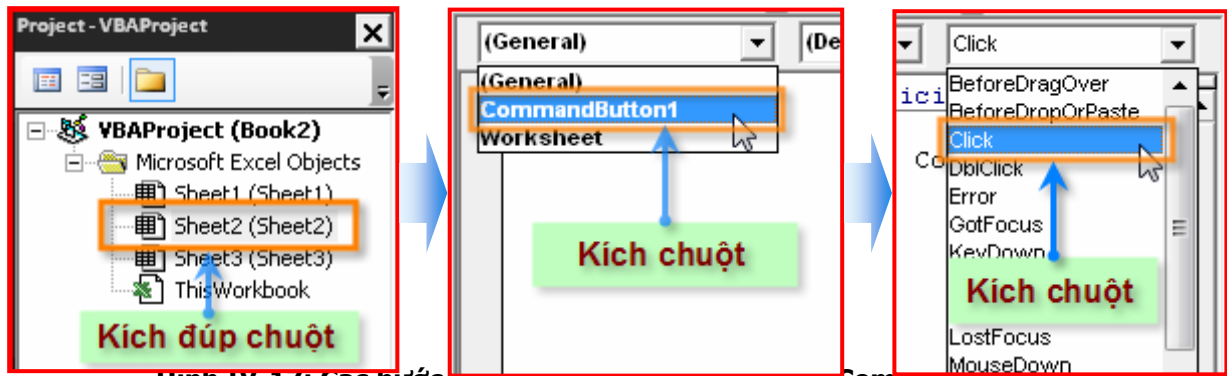
### 8.1.3. Điều khiển Command Button

Điều khiển Command Button, , thường được sử dụng khi cần người dùng thực hiện một quyết định nào đó thông qua việc kích chuột vào nút lệnh hoặc nhấn Enter tại nút lệnh. Chi tiết về các thuộc tính của điều khiển này có trong phần làm việc với Userform và các điều khiển ở phần trước.

Để cài đặt mã lệnh tương ứng khi người dùng kích chuột vào nút lệnh, sử dụng sự kiện `Click` có trong điều khiển Command Button. Các bước thực hiện như sau:

- Khởi động VBAIDE bằng cách nhấn phím `ALT+F11`.
- Trong cửa sổ Project, kích đúp chuột vào worksheet có chứa điều khiển Command Button để hiển thị cửa sổ mã lệnh của worksheet đó.
- Trong danh sách xổ xuống General ở góc trên bên trái, chọn điều khiển có tên cần thêm sự kiện, tên này chính là giá trị của thuộc tính `Name` mà ta đã gán cho điều khiển đó.
- Trong danh sách xổ xuống Declarations ở góc trên bên phải, chọn sự kiện cần cài đặt mã lệnh.
- VBAIDE sẽ tự động chèn đoạn mã lệnh khởi tạo cho sự kiện. Gõ đoạn mã lệnh cần chèn vào vị trí con trỏ đang hoạt động.





Hình IV-17: Các bước thêm sự kiện cho điều khiển Command Button.

Đoạn mã lệnh sau sẽ làm hiển thị hộp thông báo khi người dùng kích chuột vào CommandButton1:

```
Private Sub CommandButton1_Click()
    MsgBox "Ban vua nhan vao nut lenh nay"
End Sub
```

## 8.2. Các hộp thoại thông dụng

Hộp thoại là một trong những thành phần hay dùng đến nhất khi thiết kế giao diện, do đó, ngoài việc sử dụng những điều khiển để thiết kế hộp thoại trên Userform đã được đề cập ở phần trước, hoặc sử dụng chính những hộp thoại đơn giản của VB như hàm `InputBox` hoặc `MsgBox` thì người dùng có thể sử dụng những hộp thoại được Excel cung cấp và ta sẽ thấy rằng trong nhiều trường hợp, những hộp thoại này rất tiện dụng. Dưới đây là một số hộp thoại đặc trưng trong Excel.

**GỢI Ý** Tham khảo mục "Các hộp thoại thông dụng" trang 76 để biết thêm chi tiết về các hộp thoại cơ bản thường được sử dụng.

### 8.2.1. Hộp thoại `InputBox` của Excel – Hàm `InputBox`

Sử dụng hộp thoại `InputBox` (thay vì sử dụng hộp thoại `InputBox`) có nhiều ưu điểm:

- ◆ Định được kiểu dữ liệu trả về;
- ◆ Người sử dụng có thể lựa chọn một vùng dữ liệu trực tiếp trên worksheet bằng cách sử dụng chuột;
- ◆ Việc kiểm tra dữ liệu nhập vào được thực hiện tự động.

Cú pháp của hàm `InputBox` này như sau:

```
Application.InputBox(prompt, title, default, left, top, helpFile, context, type)
```

Hầu hết các tham số đều là tham số tùy chọn, chỉ có tham số `prompt` là bắt buộc phải nhập vào. Ý nghĩa của các tham số như sau:

Tham số	Giải thích
<code>prompt</code>	Đoạn văn bản sẽ hiển thị trong hộp thoại
<code>title</code>	Tiêu đề của hộp thoại <code>InputBox</code>
<code>default</code>	Giá trị mặc định. Nếu người dùng không nhập gì cả, hàm sẽ trả về giá trị mặc định này.

left, top	Toạ độ góc trên bên trái của hộp thoại.
helpFile, context	Tên tệp trợ giúp và chủ đề cần hiển thị
type	Mã xác định kiểu trả về của hàm.

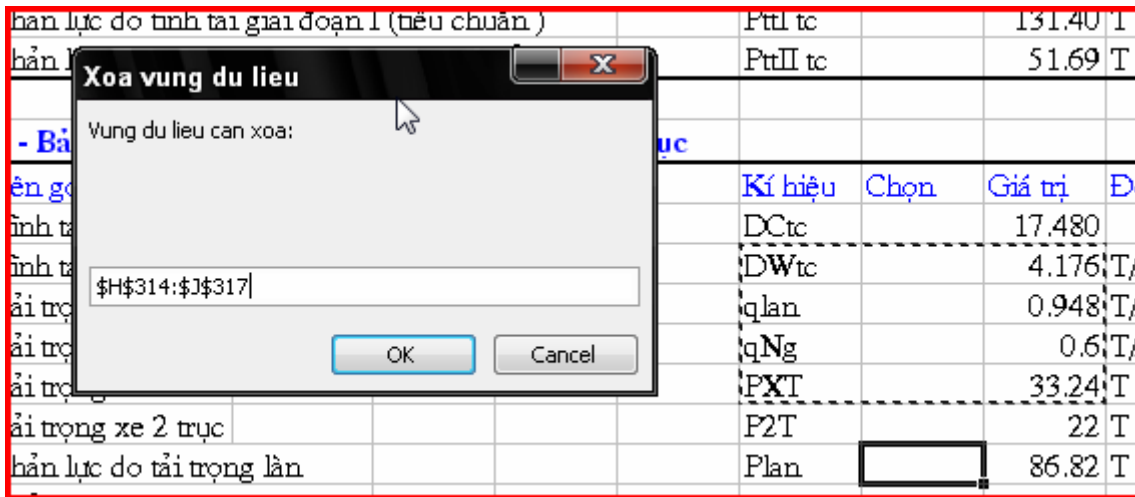
Bảng sau thể hiện các loại mã xác định kiểu trả về của hàm:

Mã	Kiểu giá trị trả về là
0	Công thức
1	Số
2	Chuỗi
4	Boolean (True hoặc False)
8	Đối tượng kiểu Range. Tham chiếu đến một vùng dữ liệu
16	Giá trị lỗi, chẳng hạn như #NA
64	Mảng giá trị

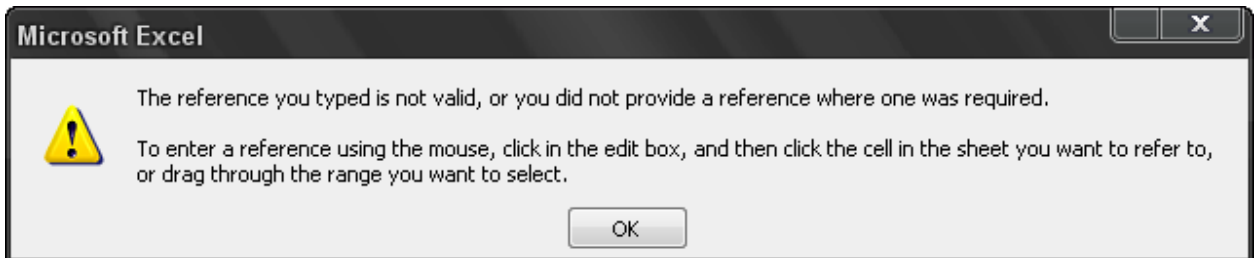
Hàm `InputBox` của Excel được sử dụng khá linh hoạt. Hàm này có thể cho phép có nhiều hơn một kiểu giá trị trả về bằng cách cộng các loại mã thích hợp. Ví dụ như muốn hộp thoại `InputBox` cho phép nhập vào cả số và chuỗi, có thể nhập tham số `type` bằng 3 (tức là 1+2, hay “Số” cộng “chuỗi”). Còn nếu gán tham số `type` bằng 8 thì người dùng có thể tự nhập vào địa chỉ của vùng dữ liệu trong hộp thoại, hoặc cũng có thể chọn vùng dữ liệu trên worksheet bằng chuột.

Ví dụ sau, thủ tục `EraseRange`, sử dụng hàm `InputBox` để người dùng lựa chọn một vùng dữ liệu để xoá. Người dùng có thể nhập vào địa chỉ của vùng dữ liệu hoặc cũng có thể chọn bằng cách dùng chuột. Hàm `InputBox` với tham số `type` bằng 8 sẽ trả về đối tượng kiểu `Range` (chú ý từ khoá `Set` trước hàm `InputBox`, vì lúc này hàm sẽ trả về đối tượng chứ không phải là một giá trị đơn thuần). Vùng dữ liệu này sẽ được xoá đi bằng phương thức `Clear`. Giá trị mặc định được hiển thị trong hộp thoại `InputBox` là địa chỉ của vùng được chọn hiện hành. Câu lệnh `On Error` nhằm mục đích sẽ thoát khỏi thủ tục khi có lỗi xảy ra.

```
Sub EraseRange()
    Dim UserRange As Range
    Dim DefaultRange As String
    DefaultRange = Selection.Address
    On Error GoTo Canceled
    Set UserRange = Application.InputBox _
        (Prompt:="Vùng dữ liệu cần xoá:", _
        Title:="Xoá vùng dữ liệu", _
        Default:=DefaultRange, _
        Type:=8)
    UserRange.Clear
    UserRange.Select
Canceled:
End Sub
```



Một lợi điểm nữa của việc sử dụng hàm `inputBox` của Excel chính là việc tự động thực hiện kiểm tra giá trị nhập vào. Trong ví dụ trên, nếu người dùng nhập vào giá trị không phải là địa chỉ của một vùng dữ liệu, Excel sẽ hiển thị một hộp thông báo và nhắc người dùng nhập lại dữ liệu.



### 8.2.2. Hộp thoại Open – Hàm `GetOpenFilename`

Trong một số chương trình, khi cần nhập vào tên tệp nào đó, ta có thể sử dụng hộp thoại `InputBox` để yêu cầu người sử dụng nhập vào tên tệp từ bàn phím. Tuy nhiên cách này có thể phát sinh lỗi do người dùng nhập vào một tên tệp không tồn tại (có thể do gõ phím sai hoặc không nhớ chính xác tên tệp). Một cách tốt hơn để làm việc này chính là sử dụng phương thức `GetOpenFilename` của đối tượng `Application`. Phương thức này sẽ hiển thị hộp thoại `Open` (giống như khi chọn trình đơn **File** ⇒ **Open**) nhưng chỉ trả về tên tệp được chọn mà không mở một tệp nào cả. Người dùng chỉ việc chọn tệp bằng các công cụ trực quan có sẵn trong hộp thoại.

Cú pháp của phương thức này như sau (tất cả các tham số đều là tham số tùy chọn):

`GetOpenFilename (FileFilter, FilterIndex, Title, ButtonText, MultiSelect)`

Tham số	Giải thích
<code>FileFilter</code>	Chuỗi chứa bộ lọc tệp.
<code>FilterIndex</code>	Số thứ tự của bộ lọc tệp mặc định.
<code>Title</code>	Tiêu đề của hộp thoại, giá trị mặc định là "Open".
<code>ButtonText</code>	Không sử dụng.
<code>MultiSelect</code>	Nếu bằng <code>TRUE</code> , người dùng có thể chọn nhiều tệp cùng một lúc. Mặc định là <code>FALSE</code> .

Tham số `FileFilter` quy định các phần tử sẽ được hiển thị trong danh sách đồ xuống "File of type" của hộp thoại. Mỗi phần tử tương ứng với một loại tệp nào đó. Phần tử là một cặp giá trị

## GIÁO TRÌNH TỰ ĐỘNG HOÁ THIẾT KẾ CẦU ĐƯỜNG

gồm tên sẽ được hiển thị trong danh sách và phần mở rộng của loại tệp tương ứng. Nếu không gán giá trị cho tham số này, giá trị mặc định sẽ là:

```
"All Files (*.*) , *.*"
```

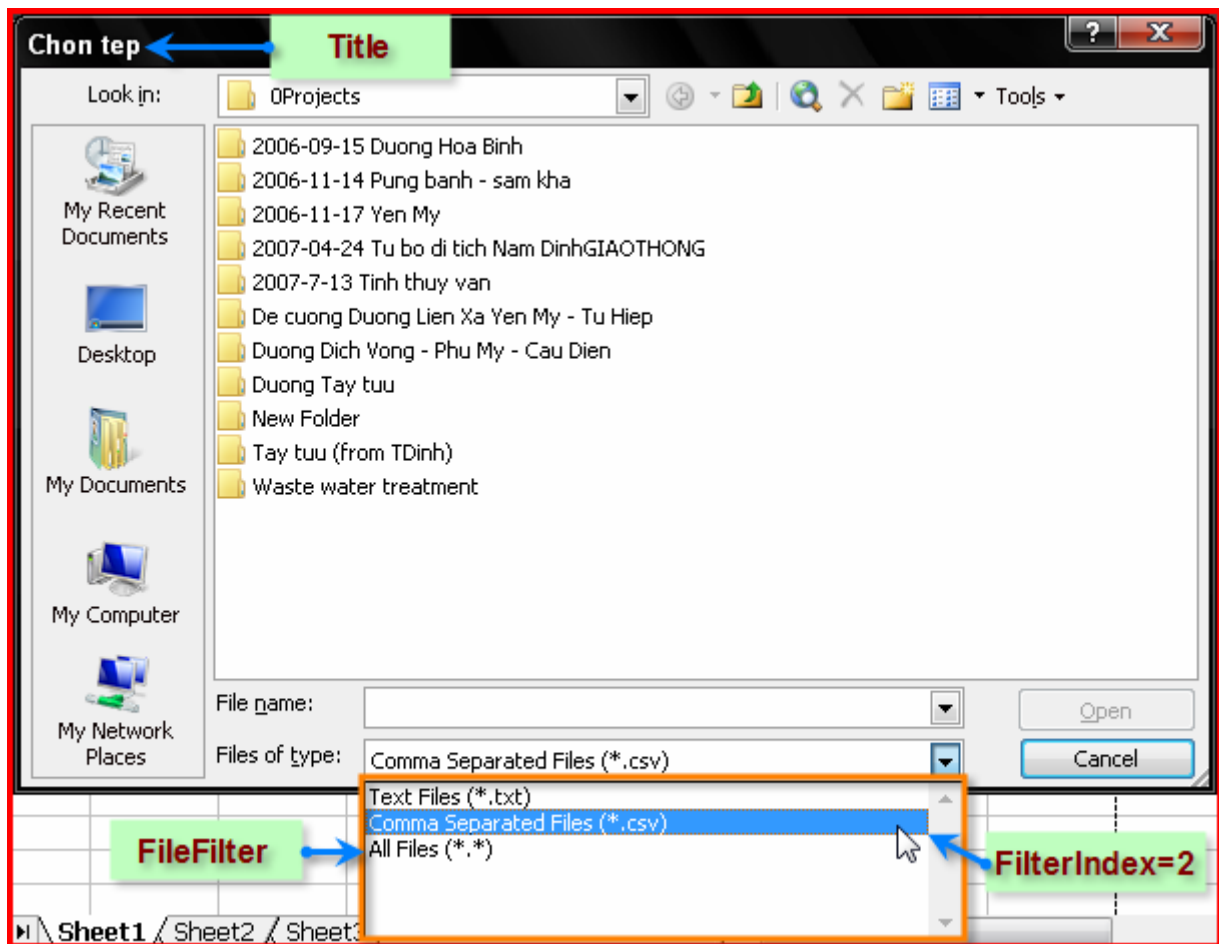
Chú ý phần đầu của chuỗi này (phần nằm phía trước dấu phẩy - All Files (\*.\*)) là đoạn văn bản sẽ được hiển thị trong danh sách. Còn phần thứ hai của chuỗi (phần nằm sau dấu phẩy - \*.\* ) chính là phần mở rộng của tệp sẽ được hiển thị.

Ví dụ sau minh họa cách tạo một chuỗi chứa trong biến Filt có thể dùng để truyền vào tham số FileFilter của phương thức GetOpenFilename. Như trong trường hợp này, người dùng có thể chọn 2 loại tệp (và một lựa chọn cho tất cả các tệp).

```
Filt = "Text Files (*.txt),*.txt," & _  
      "Comma Separated Files (*.csv),*.csv," & _  
      "All Files (*.*) , *.*"
```

Ví dụ sau sẽ nhắc người dùng chọn một tệp, sau đó sẽ hiển thị tên tệp được chọn.

```
Sub GetImportFileName()  
    Dim Filt As String  
    Dim FilterIndex As Integer  
    Dim Title As String  
    Dim FileName As String  
    ' Gán bộ lọc tệp  
    Filt = "Text Files (*.txt),*.txt," & _  
          "Comma Separated Files (*.csv),*.csv," & _  
          "All Files (*.*) , *.*"  
    ' Hiển thị các tệp *.csv là mặc định  
    FilterIndex = 2  
    ' Gán tiêu đề cho hộp thoại  
    Title = "Chon tep"  
    ' Lấy tên tệp  
    FileName = Application.GetOpenFilename _  
                (FileFilter:=Filt, _  
                FilterIndex:=FilterIndex, _  
                Title:=Title)  
    ' Thoát nếu nhấn nút Cancel  
    If FileName = "False" Then  
        MsgBox "Khong tep nao duoc chon."  
        Exit Sub  
    End If  
    ' Hiển thị tên tệp đầy đủ  
    MsgBox "Ban vua chon tep: " & FileName  
End Sub
```



Hình IV-18: Hộp thoại Open

### 8.2.3. Hộp thoại Save As – Hàm GetSaveAsFilename

Phương thức GetSaveAsFilename cũng tương tự như phương thức GetOpenFileName. Phương thức này sẽ hiển thị hộp thoại Save As, cho phép người dùng chọn hoặc chỉ định tệp để lưu, sau đó sẽ trả về tên tệp đầy đủ nhưng không thực hiện thao tác lưu nào cả.

Cú pháp của phương thức này như sau:

**GetSaveAsFilename (InitialFilename, FileFilter, FilterIndex, Title, Button Text)**

Tham số	Giải thích
InitialFilename	Xác định tên tệp gợi ý ban đầu
FileFilter	Chuỗi chứa bộ lọc tệp.
FilterIndex	Số thứ tự của bộ lọc tệp mặc định.
Title	Tiêu đề của hộp thoại, giá trị mặc định là "Save As".
ButtonText	Không sử dụng.

Ví dụ sau sẽ hiển thị hộp thoại Save As để người dùng nhập vào tên tệp, sau đó sẽ hiển thị tên tệp được lựa chọn.

```
Sub SaveAs()  
Dim fileSaveName As String
```

```
fileSaveName = Application.GetSaveAsFilename( _  
    InitialFileName:="TenTep", _  
    FileFilter:="Excel Workbook (*.xls), *.xls", _  
    Title:="Luu tap tin")  
If fileSaveName <> "False" Then  
    MsgBox "Save as " & fileSaveName  
End If  
End Sub
```

### 8.2.4. Hộp thoại chọn thư mục – Đối tượng FileDialog

Nếu cần người dùng chọn một thư mục để thực hiện thao tác nào đó, có thể thực hiện theo nhiều cách khác nhau, nhưng đơn giản nhất vẫn là sử dụng đối tượng FileDialog.

Ví dụ sau hiển thị một hộp thoại cho phép người dùng chọn thư mục. Sau đó hiển thị tên thư mục bằng cách sử dụng hộp thoại MsgBox. Tên tệp sẽ được truy cập thông qua thuộc tính SelectedItems của đối tượng FileDialog.

```
Sub GetAFolder()  
    With Application.FileDialog(msoFileDialogFolderPicker)  
        .InitialFileName = Application.DefaultFilePath & "\"  
        .Title = "Please select a location for the backup"  
        .Show  
        If .SelectedItems.Count = 0 Then  
            MsgBox "Canceled"  
        Else  
            MsgBox .SelectedItems(1)  
        End If  
    End With  
End Sub
```

Đối tượng FileDialog cho phép chỉ định thư mục ban đầu bằng cách gán giá trị cho thuộc tính InitialFileName. Trong ví dụ trên đã sử dụng thư mục mặc định của Excel làm thư mục ban đầu.

### 8.2.5. Các hộp thoại mặc định trong Excel – Tập đối tượng Dialogs

Tập đối tượng Dialogs của đối tượng Application bao gồm 258 phần tử thể hiện hầu hết các hộp thoại mặc định trong Excel. Mỗi hộp thoại có một hằng số được định nghĩa trước giúp người dùng có thể xác định được hộp thoại cần hiển thị một cách dễ dàng. Chẳng hạn như khi muốn hiển thị hộp thoại GoTo của Excel, sử dụng hằng số xlDialogFormulaGoto.

Sử dụng phương thức Show để hiển thị các hộp thoại. Ví dụ sau sẽ hiển thị hộp thoại Go To của Excel.

```
Application.Dialogs(xlDialogFormulaGoto).Show
```

Người lập trình còn có thể viết mã lệnh để kiểm tra cách thức đóng hộp thoại. Trong ví dụ sau, biến Result sẽ trả về TRUE nếu người dùng kích chuột vào nút OK, và FALSE nếu kích chuột vào nút Cancel hoặc nhấn phím ESC.

```
Result = Application.Dialogs(xlDialogFormulaGoto).Show
```

Cần phải lưu ý rằng, tính năng này không được trình bày rõ ràng trong các tài liệu trợ giúp của Excel. Các tài liệu trợ giúp cho phần này rất sơ lược, không đề cập đến một sự thật: các hộp thoại hiển thị khi gọi bằng VBA không hoàn toàn giống như khi gọi thông qua trình đơn trong

Excel. Chính vì vậy, chẳng có cách nào khác ngoài việc thử nghiệm để kiểm tra hoạt động của các hộp thoại.



**CHÚ Ý** Các hộp thoại hiển thị khi gọi bằng VBA không hoàn toàn giống như khi gọi thông qua trình đơn trong Excel.

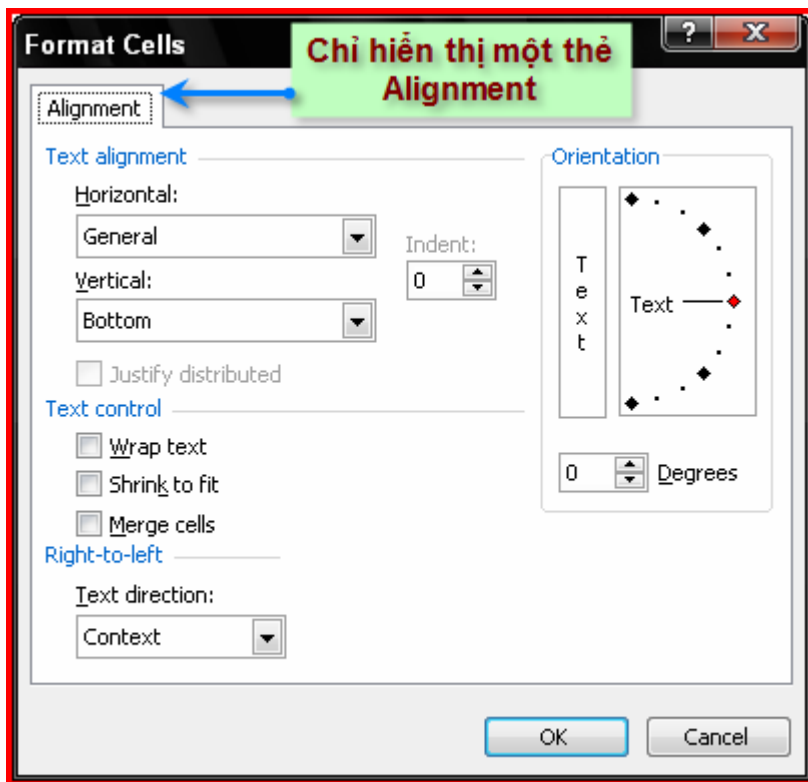
Trong trường hợp của hộp thoại Go To, khi hiển thị bằng VBA, nút Special bị mờ đi, không hoàn toàn giống như khi chọn từ trình đơn **Edit**⇒**Go To**.



Ngoài ra, còn có một vấn đề khác nữa, đó là việc không thể hiển thị được các hộp thoại có nhiều thẻ khác nhau. Lấy ví dụ với hộp thoại Format Cell, không có cách nào để hiển thị đầy đủ hộp thoại này với nhiều thẻ khác nhau từ VBA, thay vào đó, chỉ có thể hiển thị một thẻ tại một thời điểm. Đoạn mã sau chỉ hiển thị được thẻ Alignment của hộp thoại Format Cells:

```
Application.Dialogs(xlDialogAlignment).Show
```

Để hiển thị các thẻ khác trong hộp thoại Format Cells, phải sử dụng riêng lẻ các hằng số đã được định nghĩa trước như: `xlDialogFormatNumber`, `xlDialogBorder`, `xlDialogCellProtection`, `xlDialogPatterns`, hoặc `xlDialogFontProperties`.



Có rất nhiều hằng số được định nghĩa trước tương ứng với một hộp thoại trong Excel. Danh sách đầy đủ các hằng số này có thể được tra cứu với Object Browser:

1. Trong mô đun mã lệnh VBAIDE, nhấn F2 để khởi động Object Browser.
2. Trong cửa sổ Object Browser, chọn Excel ở danh sách phía trên.
3. Trong danh sách thứ 2, gõ vào xlDialog.
4. Kích chuột vào biểu tượng hình ống nhòm.



**CHÚ Ý** Hiển thị một hộp thoại không đúng ngữ cảnh sẽ làm phát sinh lỗi. Ví dụ như khi đang chọn một chuỗi số liệu trong một biểu đồ mà lại hiển thị hộp thoại Fonts (hằng số xlDialogFontProperties) thì sẽ xuất hiện thông báo lỗi bởi vì hộp thoại này xuất hiện trong tình huống này là không thích hợp.

Dưới đây là danh số một số hằng số hay được sử dụng:

Hằng số	Mô tả
xlDialogOpen	Hộp thoại Open
xlDialogSaveAs	Hộp thoại Save As
xlDialogPageSetup	Hộp thoại Page Setup
xlDialogPrint	Hộp thoại Print
xlDialogPrinterSetup	Hộp thoại Printer Setup



**GỢI Ý** Các hằng số tương ứng khi sử dụng tập đối tượng Dialogs được bắt đầu bằng xlDialog và tiếp theo là tên của hộp thoại (viết liền nhau). Trong VBAIDE, gõ xlDialog sau đó nhấn CTRL+Space để hiển thị cửa sổ gợi ý mã lệnh, trong đó sẽ có danh sách đầy đủ các hằng số liên quan.

### 8.2.6. Thực thi mục trình đơn Excel từ VBA



Một cách khác nữa để hiển thị các hộp thoại mặc định là thực thi trực tiếp thông qua trình đơn. Điều này cũng tương đương như khi sử dụng chuột để chọn một mục trình đơn trong thanh trình đơn của Excel.

Đoạn mã lệnh sau tương đương với việc người dùng chọn trình đơn **Edit⇒Go To** trực tiếp trong Excel:

```
Application.CommandBars("Worksheet Menu Bar").  
Controls("Edit").Controls("Go To...").Execute
```

Câu lệnh trên, khi thực thi sẽ hiển thị hộp thoại Go To. Cần chú ý rằng, đoạn văn bản nằm trong dấu ngoặc phải giống hệt như những gì hiển thị trên thanh trình đơn (bao gồm cả dấu ba chấm sau chữ "Go To").

Việc thực thi mục trình đơn như thế này được thực hiện khá đơn giản, hơn nữa còn khắc phục được nhược điểm không hiển thị hộp thoại có nhiều thẻ như đã đề cập ở phần "Các hộp thoại mặc định trong Excel – Tập đối tượng Dialogs" trang 166. Ví dụ sau sẽ hiển thị hộp thoại Format Cells với đầy đủ các thẻ định dạng.

```
Application.CommandBars("Worksheet Menu Bar").  
Controls("Format").Controls("Cells...").Execute
```

Ngoài ra, theo cách này, người lập trình có thể thực thi bất kỳ một mục trình đơn nào có trong thanh trình đơn của Excel.

### 8.3. Hộp thoại tùy biến – UserForm

Khi các hộp thoại mặc định trong Excel không đáp ứng được nhu cầu, người lập trình Excel có thể tạo ra các hộp thoại tùy biến của riêng mình thông qua các UserForm. Với khả năng tùy biến cao, người lập trình có thể sử dụng UserForm và các điều khiển trên đó để tạo ra những hộp thoại với nhiều tính năng hơn, phù hợp hơn với nhu cầu thực tế hơn. Việc tạo các hộp thoại tùy biến được thực hiện dễ dàng và hơn nữa với khả năng của mình, người lập trình có thể tạo ra các hộp thoại trông chẳng khác gì hộp thoại của chương trình Excel.

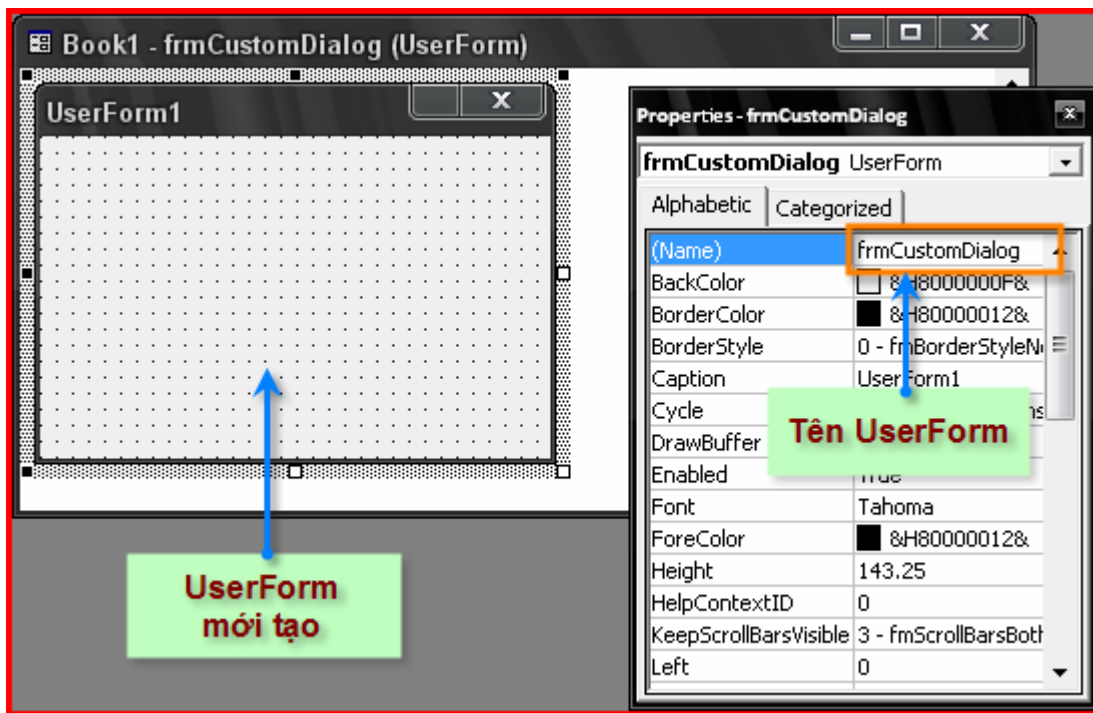
Hộp thoại tùy biến được tạo ra dựa trên UserForm thông qua VBAIDE. Thông thường, có thể tạo hộp thoại tùy biến theo các bước sau:

1. Tạo mới một UserForm vào trong dự án VBA của Workbook.
2. Viết thủ tục để hiển thị UserForm. Thủ tục này phải được đặt trong một mô-đun của VBA (chứ không phải đặt trong mô-đun của UserForm)
3. Chèn thêm các điều khiển cần thiết trên UserForm.
4. Điều chỉnh các điều khiển vừa thêm.
5. Viết mã lệnh cho các sự kiện tương ứng của các điều khiển (nếu cần). Các thủ tục này phải được đặt trong mô-đun của chính UserForm đó.

Sau khi thực hiện xong các bước trên, mỗi khi cần hiển thị hộp thoại tùy biến, chỉ cần thực thi thủ tục đã tạo ở bước 2.

#### 8.3.1. Tạo mới UserForm

Để tạo mới UserForm, khởi động VBAIDE (nhấn phím ALT+F11), chọn dự án ứng với workbook cần thêm Userform, sau đó chọn trình đơn **Insert⇒UserForm**. Các UserForm sẽ được tự động đặt tên UserForm1, UserForm2,... Người lập trình có thể thay đổi tên của UserForm để dễ dàng nhận dạng UserForm thông qua cửa sổ Properties (chọn UserForm và nhấn phím F4 để hiển thị cửa sổ Properties).



Mỗi một workbook có thể chứa nhiều UserForm khác nhau, mỗi UserForm chính là một hộp thoại tùy biến.

### 8.3.2. Hiện thị UserForm

Để hiện thị UserForm, sử dụng phương thức Show của đối tượng UserForm. Phương thức này có cú pháp như sau:

`object.Show modal`

Trong đó:

- ♦ object: là đối tượng kiểu UserForm;
- ♦ modal: là tham số tùy chọn, xác định kiểu hiện thị của UserForm. Modal có thể là một trong hai giá trị vbModal hoặc vbModeless. Nếu là vbModal, người dùng phải đóng UserForm mới có thể tiếp tục thao tác với Excel. Nếu là vbModeless, người dùng vẫn có thể vừa thao tác trên UserForm, vừa thao tác trên Excel. Mặc định là giá trị vbModal.

Đoạn mã sau sẽ hiện thị UserForm có tên là UserForm1 ở chế độ Modal:

```
UserForm1.Show
```

Ngoài ra, còn có một kỹ thuật khác để hiện thị UserForm: sử dụng phương thức Add của tập đối tượng UserForm, sau đó sử dụng phương thức Show để hiện thị UserForm. Phương thức này thích hợp khi trong dự án có nhiều UserForm và người có thể chỉ định sự xuất hiện của một UserForm bất kỳ. Đoạn mã sau sẽ hiện thị UserForm có tên là UserForm1:

```
MyForm = "UserForm1"  
UserForms.Add(MyForm).Show
```



**CHÚ Ý** Thủ tục để hiện thị hộp thoại tùy biến (UserForm) phải được đặt trong một mô-đun chuẩn của VBA (chứ không phải đặt trong mô-đun của UserForm).

VBA còn có lệnh `Load`. Lệnh này chỉ tải UserForm vào trong bộ nhớ mà không hiển thị cho đến khi sử dụng phương thức `Show` của UserForm đó. Để tải UserForm1 vào bộ nhớ, thực hiện như sau:

```
Load UserForm1
```

Khi có một UserForm tương đối phức tạp (có nhiều thành phần điều khiển cùng với nhiều dữ liệu bên trong đó), nếu sử dụng lệnh `Load` để tải UserForm vào bộ nhớ thì UserForm sẽ được hiển thị nhanh hơn khi sử dụng phương thức `Show`. Tuy nhiên, trong đại đa số các trường hợp, chỉ cần sử dụng phương thức `Show`, bởi lẽ phương thức này cũng đã tự động thực hiện lệnh `Load` (nếu UserForm chưa được tải vào bộ nhớ) ngay trước khi hiển thị UserForm.

Một khi đã được hiển thị, UserForm sẽ luôn tồn tại trên màn hình cho đến khi người dùng đóng nó lại. Vì vậy, thông thường, người lập trình sẽ tạo thêm một nút lệnh (Command Button) trên UserForm để thực hiện thủ tục đóng UserForm. Thủ tục này có thể sử dụng lệnh `Unload` để dỡ bỏ UserForm khỏi bộ nhớ của máy tính, hoặc sử dụng phương thức `Hide` của đối tượng UserForm để tạm thời ẩn UserForm.

Đoạn mã sau sẽ đóng cửa sổ UserForm1:

```
UserForm1.Hide
```

Hoặc có thể sử dụng đoạn mã sau để đóng cửa sổ UserForm1:

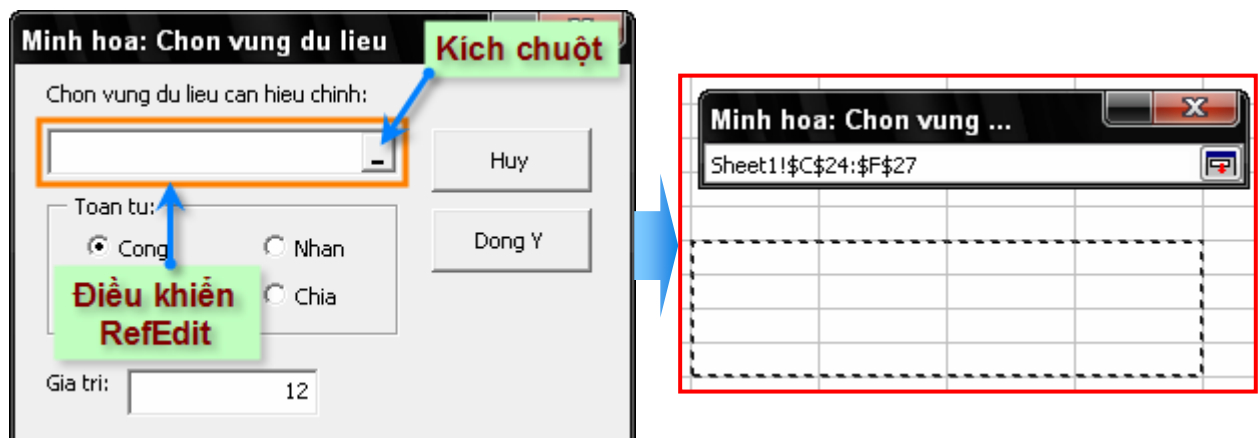
```
Unload UserForm1
```

Phương thức `Hide` chỉ tạm thời ẩn UserForm, bản thân UserForm vẫn còn trong bộ nhớ, các thuộc tính của UserForm vẫn có thể được truy cập bình thường. Còn lệnh `Unload` thì sẽ gỡ bỏ UserForm ra khỏi bộ nhớ, lúc này các thuộc tính của UserForm sẽ không thể truy cập được nữa.

### 8.3.3. Các điều khiển trên UserForm

Người lập trình có thể dùng rất nhiều loại điều khiển khác nhau lên UserForm. Thông tin chi tiết, xem lại mục “*Làm việc với UserForm và các thành phần điều khiển*” trang 60. Dưới đây chỉ trình bày thêm một điều khiển riêng của Excel, điều khiển `RefEdit`.

Điều khiển `RefEdit` cho phép người dùng lựa chọn một vùng dữ liệu bằng cách nhập địa chỉ hoặc nhập tên vùng dữ liệu hoặc sử dụng chuột để chọn trực tiếp trong worksheet. Khi người dùng kích chuột vào biểu tượng nhỏ ở góc phải của điều khiển, hộp thoại sẽ tạm thời được ẩn đi và một cửa sổ nhỏ để người dùng chọn vùng dữ liệu sẽ được hiện lên, giống hệt như các hộp thoại mặc định của Excel.



## Hình IV-19: Điều khiển RefEdit

Điều khiển RefEdit cũng tương tự như điều khiển Textbox, vì vậy có thể tham khảo thêm về điều khiển TextBox để biết thông tin về các phương thức và thuộc tính của điều khiển RefEdit.

Khi thực hiện các thao tác sử dụng RefEdit, cần ghi nhớ những điểm sau:

- ◆ Điều khiển RefEdit trả về chuỗi chứa địa chỉ của vùng dữ liệu. Sau đó, có thể chuyển chuỗi đó thành đối tượng kiểu Range sử dụng đoạn mã tương tự như sau:

```
Dim UserRange As Range  
Set UserRange = Range(RefEdit1.Text)
```

- ◆ Nên khởi tạo giá trị ban đầu cho điều khiển RefEdit bằng địa chỉ của vùng dữ liệu hiện hành. Để làm được như vậy, trong sự kiện UserForm\_Initialize của UserForm cần thêm đoạn mã lệnh tương tự như sau:

```
RefEdit1.Text = ActiveWindow.Selection.Address
```

- ◆ Đừng bao giờ nghĩ rằng RefEdit luôn trả về địa chỉ đúng. Bởi lẽ không phải chỉ có mỗi cách chọn vùng dữ liệu bằng chuột, người dùng còn có thể gõ và hiệu chỉnh địa chỉ hiển thị trên điều khiển RefEdit. Vì vậy, phải luôn kiểm tra tính đúng đắn của địa chỉ vùng dữ liệu. Đoạn mã sau minh họa cách kiểm tra lỗi này. Nếu vùng dữ liệu nhập vào không đúng, một hộp thông báo sẽ hiện lên, và cho phép người dùng nhập lại:

```
On Error Resume Next  
Set UserRange = Range(RefEdit1.Text)  
If Err <> 0 Then  
    MsgBox "Invalid range selected"  
    RefEdit1.SetFocus  
    Exit Sub  
End If  
On Error GoTo 0
```

- ◆ Người dùng có thể chọn một sheet khác trên thẻ chứa các sheet khi đang chọn vùng dữ liệu. Vì vậy, không nên giả sử rằng vùng dữ liệu được chọn sẽ nằm trên sheet hiện hành. Tuy nhiên, nếu người dùng chọn một sheet khác, địa chỉ của vùng dữ liệu sẽ được tự động thêm vào một tiền tố là tên của sheet được chọn. Chẳng hạn như: Sheet2!\$A\$1:\$C\$4

- ◆ Nếu chỉ cần lấy địa chỉ của một ô trong vùng dữ liệu mà người dùng đã chọn, người lập trình có thể chọn ra một ô ở góc trên bên trái của vùng dữ liệu đó bằng cách sử dụng đoạn mã lệnh như sau:

```
Set OneCell = Range(RefEdit1.Text).Range("A1")
```



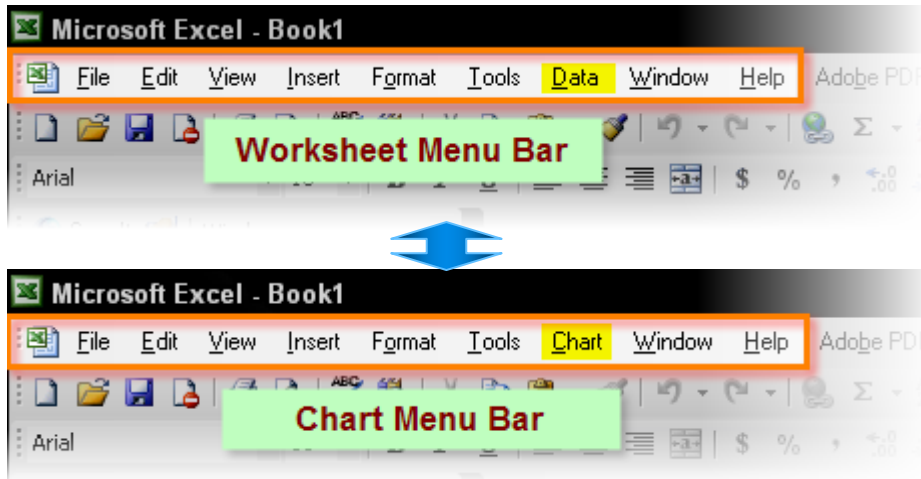
**GỢI Ý** Để người dùng chọn một vùng dữ liệu nào đó, có thể sử dụng hộp thoại InputBox của Excel, xem thêm mục "Hộp thoại InputBox của Excel – Hàm InputBox" trang 161.

## 8.4. Thao tác trên thanh trình đơn

Hầu hết các chương trình chạy trong hệ điều hành Windows đều có hệ thống thanh trình đơn bởi tính tiện dụng và hệ thống của nó. Thông qua thanh trình đơn, các chức năng của chương trình được tổ chức và liệt kê giúp người sử dụng có thể dễ dàng truy cập đến từng tính năng của chương trình một cách có hệ thống.

Đối với các ứng dụng mở rộng viết bằng VBA, việc thực thi một Macro nào đó đều được thực hiện thông qua trình quản lý Macro hoặc được thực thi trực tiếp trong VBAIDE. Điều này gây ra nhiều khó khăn cho những người dùng và làm giảm tính chuyên nghiệp của ứng dụng. Thay vào đó, với một số đoạn mã lệnh đơn giản, người lập trình có thể tự xây dựng hệ thống trình đơn, tạo nên một giao diện người dùng có tính hiệu quả cao cho ứng dụng mở rộng của mình.

Excel có hai hệ thống thanh trình đơn tương ứng với kiểu sheet được chọn là Worksheet hay Chartsheet. Thanh trình đơn thứ nhất, có tên là Worksheet Menu Bar, được hiển thị khi sheet được chọn là Worksheet hoặc khi đã đóng tất cả các Workbook. Đây là thanh trình đơn mặc định của Excel. Thanh trình đơn thứ hai, có tên là Chart Menu Bar, được hiển thị khi sheet được chọn là Chart sheet hoặc người dùng đang chọn một đối tượng Chart nhúng trong Worksheet.



Hình IV-20: Thanh trình đơn trong Excel.

### 8.4.1. Cấu trúc của hệ thống thanh trình đơn

Cấu trúc của hệ thống thanh trình đơn trong Excel có thể được thể hiện thông qua sơ đồ hình cây như sau:

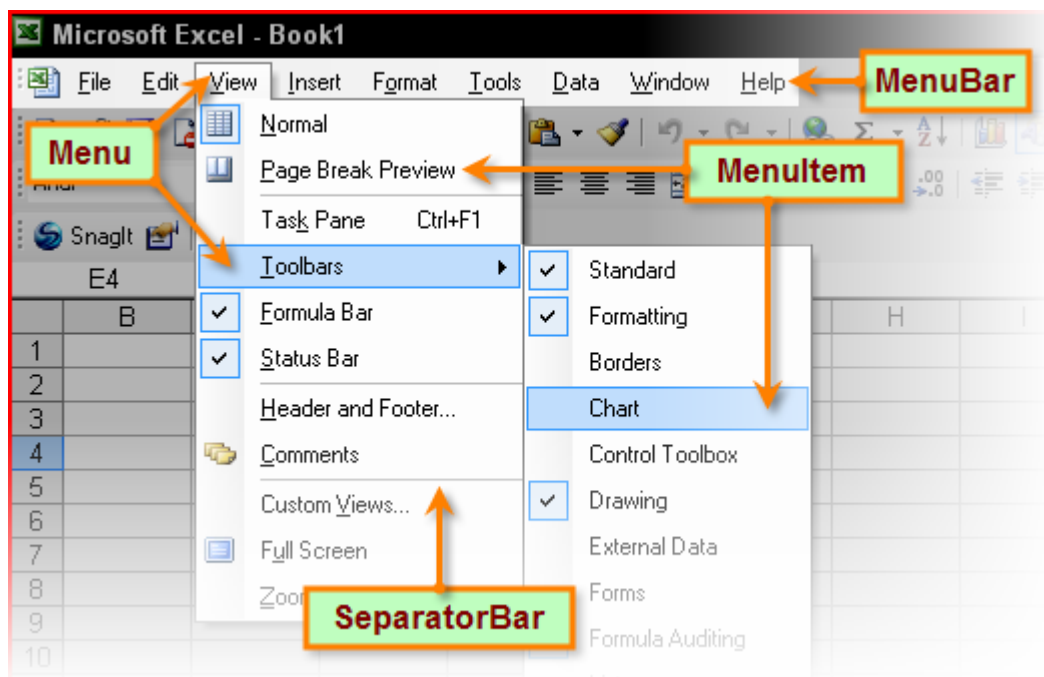


- ◆ Menu Bar: Là hàng chữ nằm ở trên cùng, ngay phía dưới thanh tiêu đề của ứng dụng Excel. Như đã đề cập, tùy vào từng ngữ cảnh mà thanh Menu Bar có thể là Worksheet Menu Bar hoặc Chart Menu Bar.
- ◆ Menu: Là một thành phần trong hệ thống trình đơn của Excel, khi người dùng kích chuột vào một Menu thì một danh sách các MenuItem sẽ hiện ra.

- ◆ Menu Item: là một thành phần của Menu xuất hiện khi người dùng kích chuột vào menu. Mỗi Menu Item sẽ thực hiện một tác vụ trong chương trình khi người dùng kích chuột lên Menu Item đó.
- ◆ Ngoài ra, trong hệ thống menu của Excel còn có khái niệm Separator Bar, là một đường gạch ngang phân cách giữa các Menu Item dùng để nhóm các Menu Item có liên quan với một mục tiêu nào đó.

Các Menu có thể lồng vào nhau theo nhiều cấp khác nhau. Một Menu cũng có thể là MenuItem nằm trong một Menu khác. Chẳng hạn như Menu View của Excel có MenuItem tên là Toolbars, và đến lượt mình, Toolbars cũng chính là một Menu, có chứa các MenuItem khác như: Standard, Formatting,...

Hình sau sẽ minh họa rõ hơn về cấu trúc của hệ thống trình đơn trong Excel.



**Hình IV-21: Hệ thống thanh trình đơn**

### 8.4.2. Tạo trình đơn tùy biến

Người lập trình có thể dễ dàng thêm và hiệu chỉnh hệ thống trình đơn trong Excel thông qua các đoạn mã lệnh bằng VBA theo các bước sau:

1. Phác thảo trình đơn cần tạo và các chức năng tương ứng.
2. Viết mã lệnh cho từng MenuItem. Mỗi đoạn mã lệnh này được chứa trong một chương trình con dạng Sub.
3. Tham chiếu đến Menu Bar, nơi cần tạo trình đơn tùy biến.
4. Tạo Menu và MenuItem.
5. Gán các đoạn mã lệnh tương ứng đã tạo ở bước 2 cho từng Menu Item.

Để tham chiếu đến Menu Bar, có thể sử dụng đoạn mã sau:

```
Dim mnuBar as CommandBar  
Set mnuBar = Application.CommandBars("Worksheet Menu Bar")
```

Để tạo Menu và MenuItem, sử dụng phương thức Add có trong tập đối tượng Controls. Thực chất, phương thức này sẽ thêm một điều khiển vào trong tập đối tượng Controls của đối tượng

gốc, nơi sẽ chứa Menu và MenuItem. Cú pháp của phương thức Add như sau (tất cả các tham số đều là tham số tùy chọn):

```
object.Add(Type, Id, Parameter, Before, Temporary)
```

Tham số	Mô tả
Object	Đối tượng cha, nơi chứa các đối tượng sẽ được thêm vào bằng phương thức Add.
Type	Xác định kiểu đối tượng sẽ được thêm vào trong tập đối tượng Controls của đối tượng Object. Tham số Type có thể bằng một trong các giá trị sau: - Nếu muốn tạo Menu: gán Type= msoControlPopup - Nếu muốn tạo Menu Item: gán Type= msoControlButton
Id	Số nguyên xác định điều khiển được xây dựng sẵn. Trong trường hợp này, khi cần tạo một đối tượng mới, có thể gán tham số này bằng 1 hoặc bỏ trống.
Parameter	Với Menu tùy biến, tham số này có thể được dùng để gửi thông tin đến các thủ tục trong Visual Basic. Thông thường, tham số này được bỏ trống.
Before	Một số xác định vị trí xuất hiện của đối tượng mới được thêm vào. Nếu tham số này được bỏ trống, đối tượng mới sẽ được thêm vào vị trí cuối cùng.
Temporary	Nếu bằng TRUE, đối tượng chỉ xuất hiện tạm thời. Nghĩa là đối tượng sẽ được xoá đi khi thoát khỏi chương trình. Giá trị mặc định của tham số này là False.

Kiểu giá trị trả về của phương thức Add là đối tượng kiểu CommandBarControl, hoặc có thể là một trong các kiểu dữ liệu sau, tùy thuộc vào giá trị của tham số Type:

- ◆ Nếu Type= msoControlPopup: kiểu giá trị trả về là CommandBarPopup.
- ◆ Nếu Type= msoControlButton: kiểu giá trị trả về là CommandBarButton.

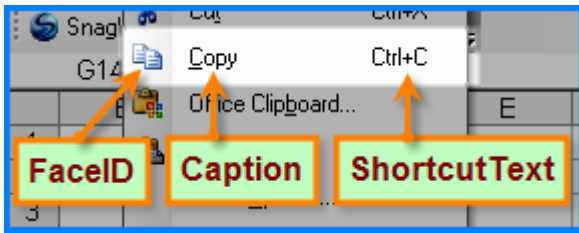
Phương thức Add chỉ tạo các đối tượng trống trên hệ thống thanh trình đơn. Vì vậy, người lập trình cần phải gán thêm các thuộc tính khác cho những đối tượng mới này.

Dưới đây là danh sách các thuộc tính của đối tượng kiểu CommandBarControl:

Thuộc tính	Mô tả
BeginGroup	Nếu gán bằng TRUE, phía trước điều khiển sẽ xuất hiện Separator Bar để ngăn cách các nhóm trình đơn.
BuiltIn	Đây là thuộc tính chỉ đọc. Trả về giá trị TRUE nếu điều khiển này là điều khiển đã được xây dựng sẵn trong Excel.
Caption	Chuỗi văn bản sẽ được hiển thị trên trình đơn.
Enabled	Nếu bằng TRUE, người dùng có thể kích chuột lên đối tượng. Nếu bằng FALSE, người dùng sẽ không thể kích chuột, và điều khiển sẽ có màu xám.
FaceID	Số nguyên thể hiện cho hình ảnh sẽ được hiển thị bên cạnh đoạn văn bản được hiển thị trên thanh trình đơn.
Id	Đây là thuộc tính chỉ đọc. Là mã số xác định các trình đơn đã được định nghĩa trước trong Excel.
OnAction	(Chỉ áp dụng với CommandBarButton) Tên của thủ tục VBA sẽ được thực thi khi người dùng kích chuột vào MenuItem.
ShortcutText	(Chỉ áp dụng với CommandBarButton) Đoạn văn bản hiển thị phần phím tắt cho MenuItem đó.

## GIÁO TRÌNH TỰ ĐỘNG HOÁ THIẾT KẾ CẦU ĐƯỜNG

State	(Chỉ áp dụng với CommandBarButton) Xác định trạng thái của MenuItem: có được nhấn hay không.
ToolTipText	Đoạn văn bản sẽ hiển thị khi người dùng trỏ chuột ngay phía trên điều khiển.
Type	Đây là thuộc tính chỉ đọc. Số nguyên xác định kiểu của điều khiển



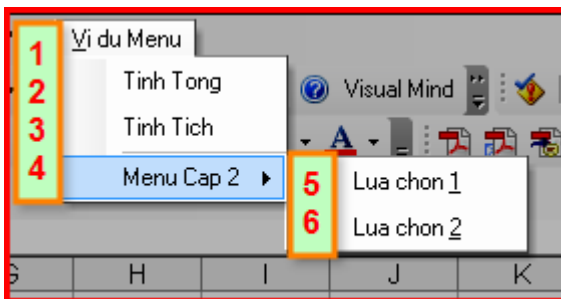
Thông thường, sau khi tạo mới Menu và MenuItem cần gán các thuộc tính sau:

- ◆ Caption
- ◆ OnAction
- ◆ FaceID

Dưới đây là một ví dụ minh họa các bước tạo mới một Menu trong thanh trình đơn “Worksheet Menu Bar”

### Ví dụ: Tạo Menu mới

1. Phác thảo cấu trúc của Menu như sau:



2. Viết mã lệnh cho từng MenuItem

```
'MÃ LỆNH CHO MENUITEM2: TINH TONG
Sub Macro1 ()
  MsgBox "Ban da chon MenuItem: Tinh Tong"
End Sub

'MÃ LỆNH CHO MENUITEM3: TINH TICH
Sub Macro2 ()
  MsgBox "Ban da chon MenuItem: Tinh Tich"
End Sub

'MÃ LỆNH CHO MENUITEM6: LUA CHON 1
Sub Macro3 ()
  MsgBox "Ban da chon MenuItem: Lua chon 1"
End Sub

'MÃ LỆNH CHO MENUITEM7: LUA CHON 2
Sub Macro4 ()
  MsgBox "Ban da chon MenuItem: Lua chon 2"
End Sub
```



3. Tạo hệ thống Menu và gán mã lệnh cho từng MenuItem

```

Sub TaoMenu()
    Dim cb As CommandBar
    Dim cpop As CommandBarPopup
    Dim cpop2 As CommandBarPopup
    Dim cbtn As CommandBarButton
    ' LẤY THAM CHIẾU ĐẾN THANH TRÌNH ĐƠN
    Set cb = Application.CommandBars("Worksheet Menu Bar")

    ' TẠO MENU1: "VI DU MENU" (CommandBarPopup).
    Set cpop = cb.Controls.Add(Type:=msoControlPopup, Temporary:=True)
    cpop.Caption = "&Vi du Menu"

    ' TẠO MENUITEM2: "TINH TONG" (CommandBarButton).
    ' (thêm MenuItem vào MENU1)
    Set cbtn = cpop.Controls.Add(msoControlButton, , , , True)
    ' Gán thuộc tính cho MenuItem.
    cbtn.Caption = "Tinh Tong"           \ Gán tiêu đề
    cbtn.OnAction = "Macro1"           \ Gán mã lệnh

    ' TẠO MENUITEM3: "TINH TICH" (CommandBarButton).
    Set cbtn = cpop.Controls.Add(msoControlButton, , , , True)
    cbtn.Caption = "Tinh Tich"
    cbtn.OnAction = "Macro2"

    ' TẠO MENU4: "MENU CAP 2" (CommandBarPopup).
    ' Đây là MenuItem bắt đầu một nhóm trình đơn khác
    Set cpop2 = cpop.Controls.Add(msoControlPopup, , , , True)
    cpop2.Caption = "Menu Cap 2"
    ' Thêm SeparatorBar vào phía trước Menu này.
    cpop2.BeginGroup = True

    ' TẠO MENUITEM5: "LUA CHON 1" (CommandBarButton).
    Set cbtn = cpop2.Controls.Add(msoControlButton, , , , True)
    cbtn.Caption = "Lua chon &1"
    cbtn.OnAction = "Macro3"

    ' TẠO MENUITEM6: "LUA CHON 2" (CommandBarButton).
    Set cbtn = cpop2.Controls.Add(msoControlButton, , , , True)
    cbtn.Caption = "Lua chon &2"
    cbtn.OnAction = "Macro4"
End Sub

```

Trong các câu lệnh tạo hệ thống trình đơn như trên, tham số Temporary của phương thức Add đều được gán bằng True, vì vậy, khi người dùng thoát khỏi Excel thì các hệ thống trình đơn vừa thêm vào sẽ được tự động xoá đi.

8.4.3. Xoá trình đơn tùy biến

Khi người dùng chỉ đóng workbook mà không đóng Excel, trình đơn vừa được thêm vào vẫn còn được hiển thị trên hệ thống thanh trình đơn của Excel hoặc người dùng làm việc với một workbook khác mà không cần đến những tính năng trong trình đơn. Như vậy, có những lúc cần phải xoá trình đơn vừa được thêm vào. Để thực hiện điều này, có thể sử dụng phương thức Delete có trong đối tượng kiểu CommandBarControl hoặc CommandBarPopup hoặc CommandBarButton.

Đoạn mã lệnh sau thực hiện xoá trình đơn “Vi du Menu” đã được tạo ra ở ví dụ trên.

```

Sub XoaMenu ()
    Dim cb As CommandBar
    Dim cbp As CommandBarPopup
    'Lấy tham chiếu đến thanh trình đơn
    Set cb = Application.CommandBars("Worksheet Menu Bar")
    On Error Resume Next
    'Tham chiếu đến trình đơn "Vi du Menu"
    Set cbp = cb.Controls("Vi du Menu")
    If Not IsNull(cbp) Then
        cbp.Delete
    End If
End Sub

```

Ngoài ra, thay vì xoá trình đơn vừa tạo ra, người lập trình có thể thiết lập lại trạng thái ban đầu của hệ thống thanh trình đơn trong Excel thông qua phương thức `Reset`. Sau khi sử dụng phương thức này, tất cả các trình đơn do người dùng tạo ra sẽ được xoá đi, và hệ thống thanh trình đơn sẽ trở về trạng thái mặc định.

```

Sub ResetMenu ()
    Dim cb As CommandBar
    Dim cbp As CommandBarPopup
    'Lấy tham chiếu đến thanh trình đơn
    Set cb = Application.CommandBars("Worksheet Menu Bar")
    cbp.Reset
End Sub

```

Trong hầu hết các trường hợp, người lập trình sẽ tạo trình đơn lúc mở workbook, và sẽ xoá trình đơn khi đóng workbook và ta có thể thực hiện tự động quá trình này thông qua việc xử lý sự kiện liên quan đến việc mở và đóng Workbook. Trong sự kiện `Workbook_Open`, gọi đến thủ tục thực hiện việc tạo trình đơn, còn trong sự kiện `Workbook_BeforeClose`, gọi đến thủ tục thực hiện việc xoá trình đơn.

```

'SỰ KIỆN Workbook_Open
Private Sub Workbook_Open ()
    'Gọi thủ tục thực hiện việc tạo trình đơn
    TaoMenu
End Sub

'SỰ KIỆN Workbook_BeforeClose
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    'Gọi thủ tục thực hiện việc xoá trình đơn
    XoaMenu
End Sub

```

#### 8.4.4. Gán phím tắt cho Menu Item

Gán phím tắt cho Menu Item thực chất là gán phím tắt cho Macro tương ứng với Menu Item đó (là Macro được gán cho Menu Item thông qua thuộc tính `OnAction`). Bổ sung đoạn mã lệnh sau vào cuối thủ tục `TaoMenu` ở ví dụ trước để gán phím tắt là `CTRL+SHIFT+T` cho Menu Item "Tinh Tong":

```

'Tạo phím tắt cho MenuItem
Application.MacroOptions _
    Macro:="Macro1", _
    HasShortcutKey:=True, _
    ShortcutKey:="T"

```

Trong khi tạo hệ thống trình đơn “Vi dụ Menu” ở ví dụ trước, Menu Item “Tinh Tong” có thuộc tính `OnAction` được gán bằng “`Macro1`”. Do đó để gán phím tắt cho Menu Item này, người lập trình phải thực hiện thông qua việc gán phím tắt cho Macro có tên là “`Macro1`”.



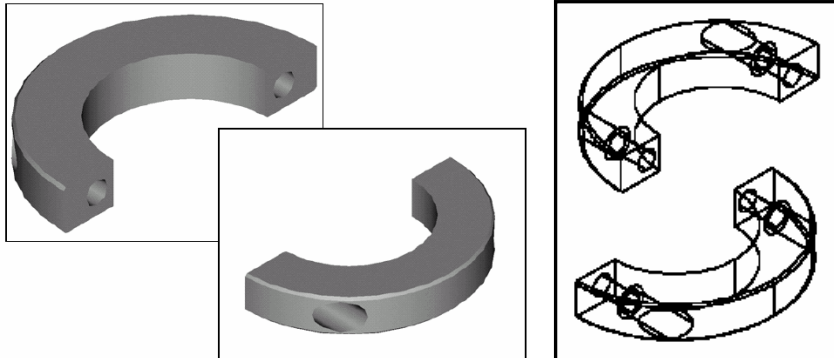
# CHƯƠNG V: LẬP TRÌNH TRÊN AUTOCAD

## 1. Tổng quan về AutoCAD

### 1.1. Khả năng của AutoCAD

AutoCAD là một phần mềm hỗ trợ tạo bản vẽ kỹ thuật được dùng phổ biến nhất hiện nay. Đây là sản phẩm của hãng Autodesk và được phát triển liên tục trong nhiều năm nay, điều này thể hiện ở việc cập nhật hàng năm của các phiên bản AutoCAD.

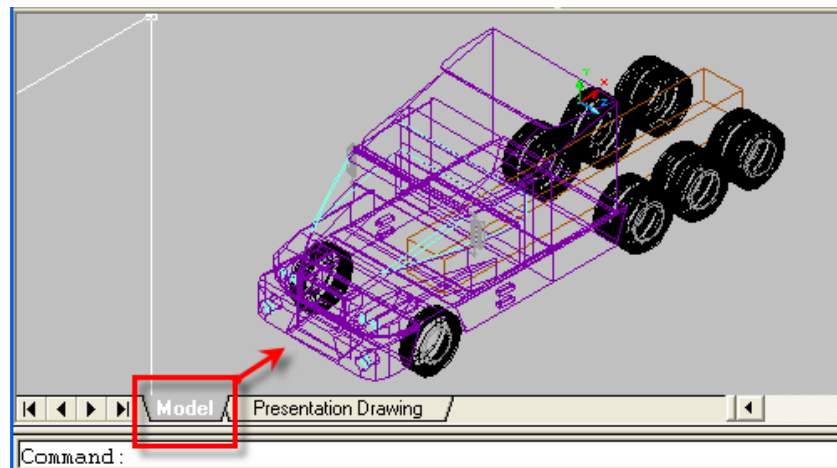
Với AutoCAD, người dùng có thể dễ dàng tạo ra bản vẽ kỹ thuật dạng 2 chiều và dựng mô hình ba chiều cho các vật thể với nhiều cách thể hiện khác nhau như dạng khung lưới hoặc dạng vật thể đặc như hình dưới.



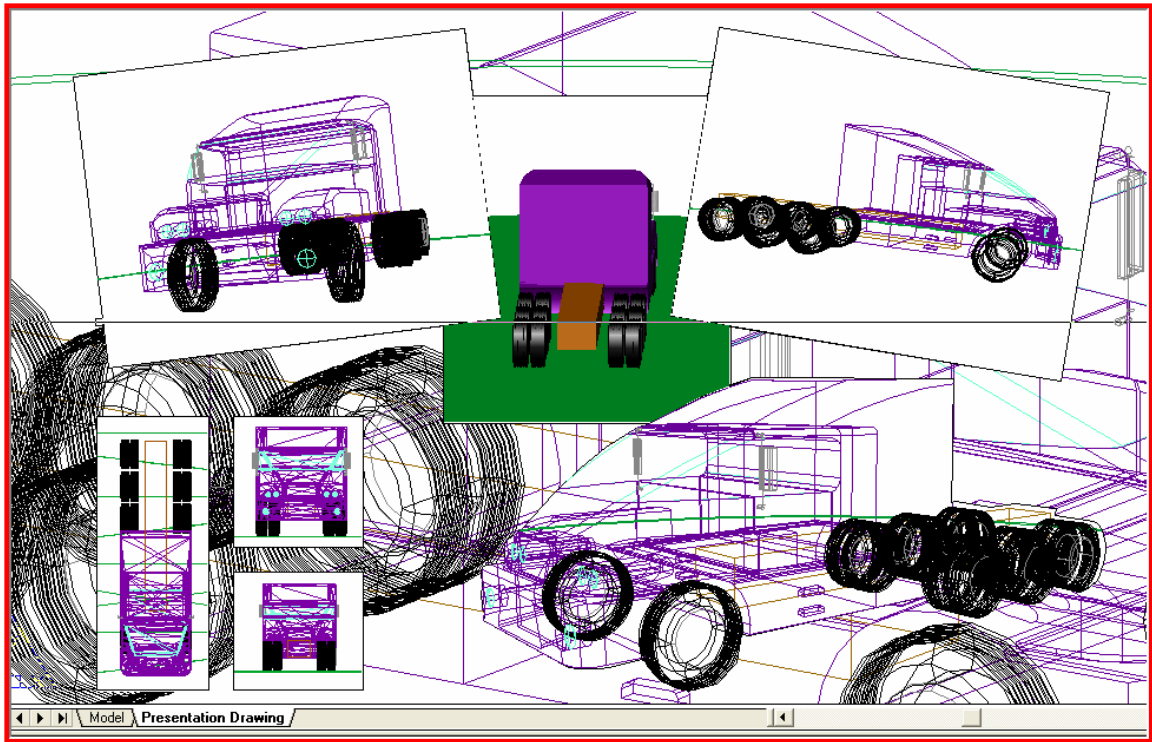
Hình vẽ trong AutoCAD được tổ chức chủ yếu theo dạng vector và chuẩn lưu trữ dạng DWG được biết đến như là chuẩn lưu trữ hình vẽ dạng vector hiệu quả nhất thế giới. Để tạo sự thuận lợi tối đa cho người dùng, AutoCAD đã được thiết kế với cấu trúc và tính năng rất hợp lý:

◆ Không gian để tạo bản vẽ được chia thành hai loại:

- Không gian mô hình (Model), là nơi mà người dùng có thể vẽ hay dựng mô hình của bất cứ vật thể nào mà không cần quan tâm đến giới hạn về kích thước của đối tượng, của bản vẽ cũng như tỷ lệ trình bày.
- Không gian trình bày hay còn gọi là không gian in (Layout), là nơi mà người dùng có thể vẽ hay dựng mô hình như không gian mô hình, nhưng đây không phải là mục đích chính của không gian in. Mục đích chính của không gian in là giúp cho người dùng có thể biểu diễn hoặc trình bày bản vẽ theo ý tưởng của mình dựa trên mô hình đã được dựng (hay đã được vẽ) trong không gian mô hình. Trong không gian in, với số lượng không hạn chế, người dùng có thể dễ dàng tạo ra những bản in có tỷ lệ khác nhau, cách bố trí, sắp đặt khác nhau từ một mô hình đã vẽ này. Hình dưới là mô hình của vật thể được xây dựng trong không gian mô hình.



Với mô hình này, khi sử dụng không gian in (Layout), ta có thể tạo ra một bản trình bày khá ấn tượng như hình sau:



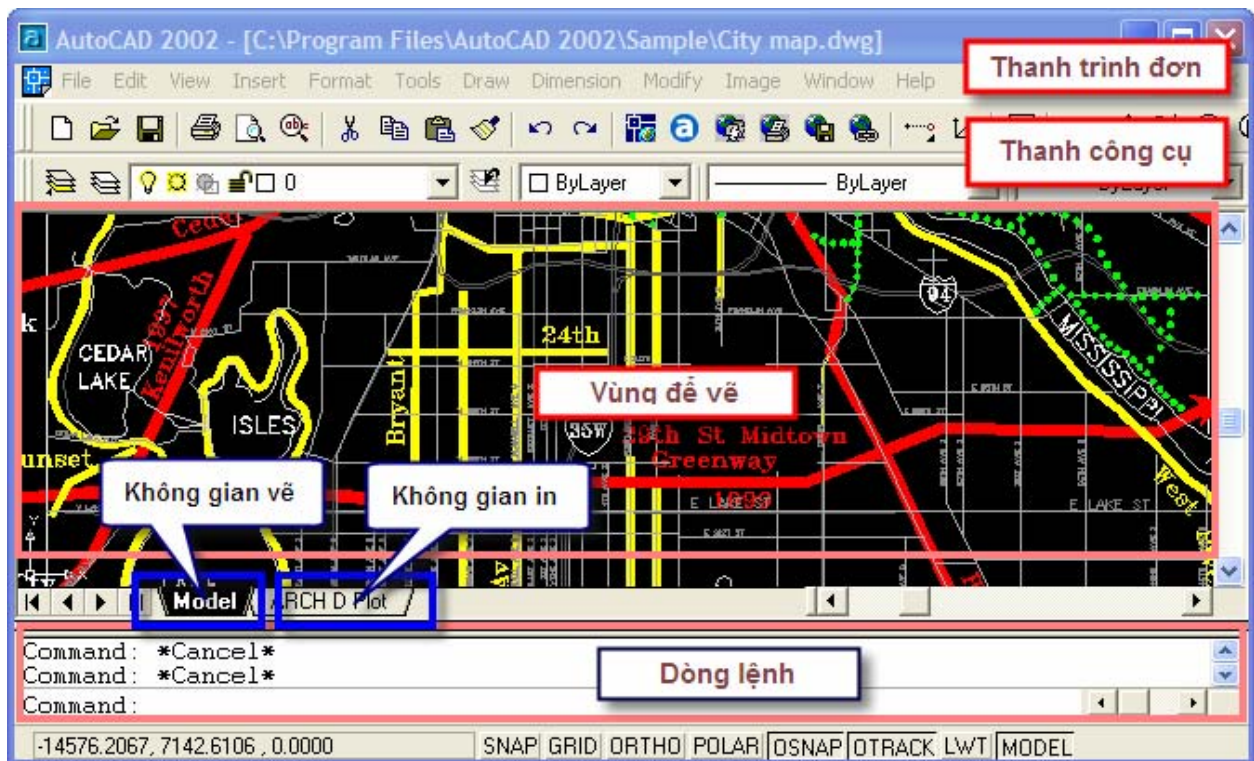
- ◆ Thao tác tạo bản vẽ được thực hiện thông qua các lệnh trong thanh trình đơn, thanh công cụ, và đặc biệt là thông qua dòng lệnh của AutoCAD. Với hàng trăm lệnh sẵn có, cùng với cách thực hiện lệnh đa dạng, cho nên người dùng có thể làm quen và sử dụng AutoCAD trong một thời gian ngắn.
- ◆ Hình vẽ trong AutoCAD, cho dù đơn giản hay phức tạp đến mấy, đều được tạo nên từ những đối tượng hình học cơ bản. Và những đối tượng hình học cơ bản này lại được một hệ thống các đối tượng phi hình học khác trong AutoCAD hỗ trợ việc tạo ra chúng.
- ◆ Với cách tổ chức các đối tượng hình học theo lớp (Layer), AutoCAD cho phép người dùng tổ chức bản vẽ, cho dù phức tạp đến mấy, thành từng lớp theo những chủ đề khác nhau, khiến cho việc quản lý và thao tác với bản vẽ trở nên dễ dàng hơn.
- ◆ Các tiện ích về in bản vẽ khiến cho việc in ấn trở nên đơn giản và chuyên nghiệp.
- ◆ Khi những tính năng sẵn có của AutoCAD không đáp ứng được nhu cầu của người dùng thì người dùng có thể sử dụng khả năng cho phép lập trình mở rộng của AutoCAD để bổ sung thêm hay tạo mới những tính năng chuyên biệt cho AutoCAD nhằm đáp ứng được nhu cầu cá nhân.

### 1.2. Giao diện của AutoCAD

Giao diện của AutoCAD, về cơ bản, là một giao diện đồ họa khá linh hoạt, bao gồm vùng để vẽ và các thành phần trợ giúp cho các thao tác vẽ. Các lệnh của AutoCAD có thể được thực hiện từ thanh trình đơn, từ thanh công cụ và từ dòng lệnh trong giao diện chính (như hình dưới). Với cách thiết kế tương tác trực quan, người dùng có thể lựa chọn hay định vị một cách linh hoạt các đối tượng trên bản vẽ, giúp cho việc vẽ được nhanh và chính xác.

Trong thanh trình đơn và thanh công cụ, các lệnh được tổ chức theo nhóm chức năng và người dùng có thể tự do thêm bớt hay thay đổi các thành phần trong các thanh này thông qua các thiết lập tùy chọn trong Customize (bấm phím phải chuột trên thanh công cụ) hoặc điều chỉnh nội dung tệp ACAD.MNU của AutoCAD. Không gian vẽ và không gian in (với hai khái niệm tương đương trong phần sau là ModelSpace và PaperSpace) được tổ chức độc lập và cho phép

tham chiếu, cùng với việc cho phép người dùng tạo đối tượng hình học trong cả hai không gian này khiến cho việc tổ chức bản vẽ đạt hiệu quả cao.



Với các lệnh, khi được gọi từ dòng lệnh, ta có thể định nghĩa lại tên của chúng để thuận tiện cho việc sử dụng của cá nhân và dễ nhớ.

### 1.3. Khả năng mở rộng của AutoCAD

Mặc dù AutoCAD được thiết kế với cấu trúc rất linh hoạt, giao diện thân thiện và dễ sử dụng, rất nhiều đối tượng hình học và phi học sẵn có, hàng trăm lệnh hỗ trợ tạo bản vẽ và điều khiển AutoCAD có sẵn đã khiến cho việc sử dụng AutoCAD nhanh, dễ và hiệu quả cao, nhưng ngay từ những phiên bản đầu tiên, AutoCAD đã được thiết kế với kiến trúc mở, nghĩa là nó cho phép người dùng tự phát triển thêm những phần mềm mới chạy trên AutoCAD, bổ sung những tính năng mới cho AutoCAD với mục đích giúp người dùng có thể biến AutoCAD thành một công cụ làm việc chuyên dụng với hiệu suất cao.

Bên trong AutoCAD, từ phiên bản 2000 (R15), đã tích hợp sẵn hai công cụ lập trình mở rộng cho AutoCAD là AutoLISP và VBA. Bên cạnh đó, nếu như người dùng có nhu cầu xây dựng những phần mềm đòi hỏi can thiệp sâu vào AutoCAD, thì ObjectARX là một lựa chọn phù hợp. Với ObjectARX, người dùng có thể sử dụng ngôn ngữ C++ trong bộ công cụ lập trình Visual Studio của Microsoft, một bộ công cụ lập trình được coi là mạnh và thân thiện nhất hiện nay, để xây dựng phần mềm. Như vậy ta có thể lập trình mở rộng AutoCAD với các công cụ sau:

- ◆ **AutoLISP và Visual LISP:** là công cụ lập trình đơn giản với ngôn ngữ lập trình là AutoLISP và môi trường lập trình Visual LISP. Một chương trình viết bằng AutoLISP được lưu trữ độc lập trên tệp văn bản và được gọi vào AutoCAD khi cần dùng đến bằng một lệnh riêng. Do đặc thù riêng của ngôn ngữ lập trình, AutoLISP chỉ thích hợp cho việc xây dựng những chương trình dạng tiện ích với quy mô nhỏ và không đòi hỏi những kỹ thuật hay thuật toán phức tạp.
- ◆ **VB và VBAIDE:** thường được gọi là VBA, là công cụ lập trình được tích hợp sẵn trong AutoCAD trên cơ sở ngôn ngữ và môi trường lập trình của Visual Basic. Chương trình VBA có thể được lưu trữ độc lập hay nhúng vào bản vẽ. Do xuất phát từ Visual Basic nên VBA trong AutoCAD là một công cụ lập trình mạnh, dễ phát triển và hiệu suất cao. Tuy

nhiên, cũng như AutoLISP, với VBA, người dùng không thể can thiệp sâu vào bên trong AutoCAD, ví dụ như không thể bổ sung thêm vào AutoCAD một đối tượng hình học hoặc phi hình học mới.

- ✦ ObjectARX và VC++: thường được gọi là ObjectARX, thực chất đây là một thư viện lập trình cho VC++, với thư viện này, người dùng có thể sử dụng VC++ để tạo ra các chương trình mà không có bất kỳ sự hạn chế nào trong việc tương tác với AutoCAD. Chương trình sẽ được biên dịch thành dạng ARX (chính là DLL) và khi cần sử dụng thì người dùng có thể gọi chúng vào trong AutoCAD bằng một lệnh riêng.

Qua kinh nghiệm triển khai các dự án phần mềm trên AutoCAD, có thể thấy rằng hầu hết các nhu cầu tính toán và tự động tạo bản vẽ thông thường đều có thể thực hiện một cách dễ dàng và nhanh chóng bằng VBA và AutoLISP, vì vậy, trong khuôn khổ của giáo trình này, nội dung lập trình mở rộng AutoCAD bằng VBA được đặt là trọng tâm bởi tính hiệu quả của nó. Bên cạnh đó, để khai thác tốt nhất AutoCAD, thì việc kết hợp VBA với AutoLISP để thực hiện một dự án phần mềm là một lựa chọn hay, do đó, trong giáo trình này cũng đề cập đến một phần sự kết hợp này. Còn đối với ObjectARX, những ai quan tâm có thể xem trong giáo trình môn học “Lập trình trên ứng dụng nền”.

## 2. Quản lý dự án VBA trong AutoCAD

### 2.1. Dự án VBA trong AutoCAD

Không giống như trong Excel, chỉ có một loại dự án VBA nhúng ngay trong tệp bảng tính XLS, các dự án VBA trong AutoCAD được phân thành 2 loại sau:

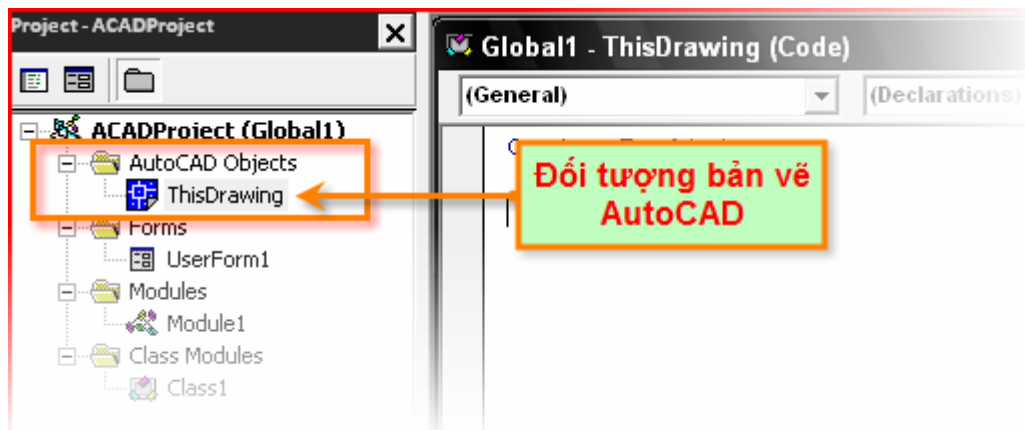
- ✦ **Dự án nhúng (Embedded Project):** là dự án VBA được lưu trữ trong tệp DWG cùng với các thông tin khác của bản vẽ trong AutoCAD. Dự án nhúng có một số đặc điểm sau:
  - Không thể đóng hoặc mở các bản vẽ AutoCAD bởi dự án loại này được thiết lập là chỉ làm việc bên trong bản vẽ chứa nó.
  - Khi sử dụng dự án nhúng, người dùng không cần phải nhớ nơi lưu trữ dự án, không cần phải thực hiện tải dự án vào AutoCAD mỗi khi cần sử dụng các chức năng có trong dự án bởi tất cả các thao tác đó đều được thực hiện tự động khi mở bản vẽ có chứa dự án nhúng.
  - Các chức năng được lập trình trong dự án VBA nhúng chỉ có hiệu lực đối với bản vẽ chứa nó, và như vậy, khi muốn sử dụng các chức năng này cho những bản vẽ khác, người sử dụng buộc phải sao chép dự án VBA đó sang các tệp bản vẽ này.
- ✦ **Dự án độc lập (Global Project):** là dự án được lưu tách biệt trong một tệp có phần mở rộng là \*.DVB và không phụ thuộc vào một bản vẽ nào cả. Dự án VBA độc lập có một số đặc điểm sau:
  - Để sử dụng một tính năng nào đó trong dự án VBA độc lập, người sử dụng phải tải dự án đó vào AutoCAD.
  - Dự án VBA độc lập có khả năng làm việc linh hoạt hơn, có khả năng đóng hoặc mở bản vẽ bất kỳ hay có thể tác động lên tất cả các bản vẽ đang mở trong phiên làm việc của AutoCAD. Với dự án độc lập, việc phân phối và chia sẻ mã lệnh được thực hiện dễ dàng hơn so với loại dự án nhúng. Dự án độc lập cũng rất thích hợp để lưu trữ, tập hợp thành bộ thư viện để sử dụng trong tất cả các bản vẽ.

Lợi điểm lớn nhất của dự án nhúng là khả năng tự động tải dự án mỗi khi mở bản vẽ. Tuy nhiên, việc sử dụng dự án nhúng cũng ẩn chứa nhiều điểm bất lợi, chẳng hạn như kích thước của tệp bản vẽ có chứa dự án nhúng sẽ tăng lên do phải chứa thêm cả phần dự án VBA; hoặc khi muốn hiệu chỉnh dự án VBA, người lập trình phải tìm lại tất cả các tệp bản vẽ có chứa dự án nhúng để thay đổi cho từng tệp một, một công việc nhàm chán!



Tuy không có được khả năng tự động tải vào như dự án nhúng, nhưng loại dự án độc lập thường được sử dụng nhiều hơn bởi các thao tác tải một dự án VBA vào trong AutoCAD thường rất đơn giản và nhanh chóng. Chỉ có một điểm lưu ý là khi làm việc với bản vẽ ở nhiều máy tính khác nhau thì cần phải mang theo cả tệp bản vẽ lẫn tệp dự án VBA độc lập, còn với dự án VBA nhúng thì ta chỉ cần mang theo tệp bản vẽ là đủ.

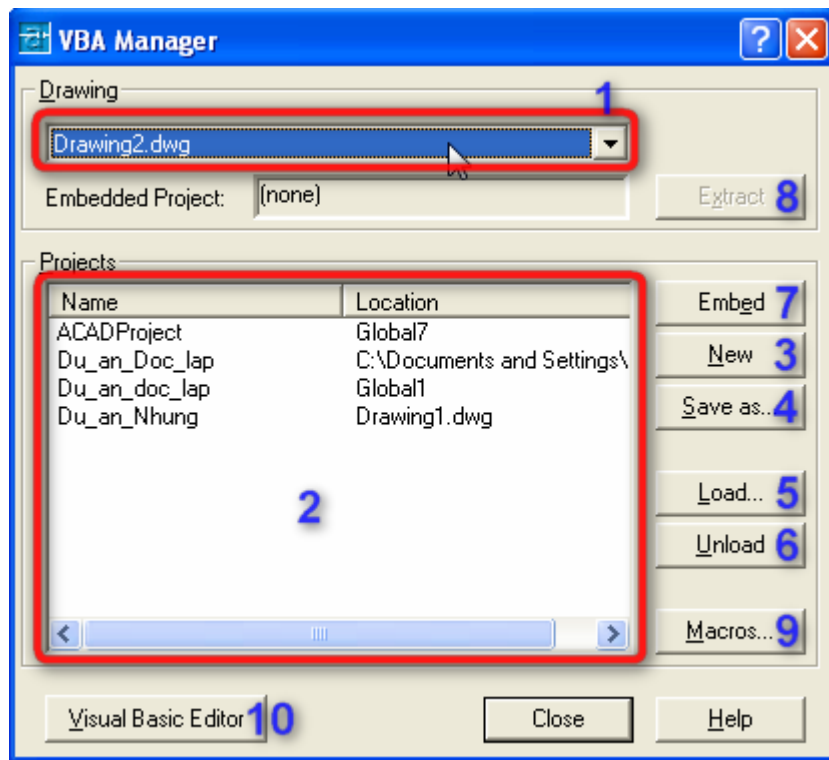
Cấu trúc của một dự án VBA trong AutoCAD cũng tương tự như trong Excel. Ngoài các thành phần như UserForm, mô-đun chuẩn, mô-đun lớp, dự án VBA trong AutoCAD còn có một thành phần khác là: AutoCAD Objects – Các đối tượng của AutoCAD. Trong thành phần này có chứa mô-đun ThisDrawing, đây chính là một đối tượng đại diện cho bản vẽ hiện hành của AutoCAD. Như vậy trong một dự án VBA của AutoCAD có 4 thành phần có thể chứa mã lệnh là: Userform, mô-đun chuẩn, mô-đun ThisDrawing và mô-đun lớp.



**Hình V-1: Thành phần AutoCAD Objects và mô-đun ThisDrawing của dự án VBA trong AutoCAD**

## 2.2. Trình quản lý dự án VBA

Với AutoCAD, quá trình quản lý các dự án VBA được thực hiện rất dễ dàng thông qua trình quản lý dự án VBA – VBA Manager. Để hiển thị cửa sổ VBA Manager, chọn trình đơn **Tool**⇒**Macro**⇒**VBA Manager...** (hoặc gọi lệnh `VBAMAN` từ dòng lệnh của AutoCAD).



**Hình V-2: Trình quản lý dự án VBA – VBA Manager.**

Ý nghĩa của từng thành phần trong cửa sổ VBA Manager:

1. Danh sách các bản vẽ hiện có trong AutoCAD (sử dụng đến danh sách này khi muốn nhúng một dự án nào đó vào trong một bản vẽ hoặc tách dự án ra khỏi bản vẽ đó).
2. Danh sách các dự án VBA hiện đang được tải trong VBAIDE. Các dự án VBA, khi đã được tải vào VBAIDE (các dự án có trong danh sách) thì người dùng có thể sử dụng các tính năng có trong dự án, hiệu chỉnh dự án và tạo thêm các tính năng mới khi cần.
3. Tạo một dự án VBA mới, mặc định, dự án VBA mới được tạo là dự án độc lập
4. Lưu dự án VBA với tên khác (chỉ có hiệu lực với các dự án độc lập). Dự án được lưu với tên khác này là dự án được chọn trong danh sách 2.
5. Tải một dự án độc lập vào VBAIDE. Người dùng sẽ được yêu cầu chọn một tệp dự án VBA (\*.DVB) để tải vào VBAIDE.
6. Đóng dự án độc lập khỏi VBAIDE. Người dùng không thể truy cập đến các thành phần trong dự án nữa (tuy nhiên tệp chứa dự án đó vẫn còn tồn tại trong máy tính).
7. Nhúng một dự án VBA vào một bản vẽ định trước. Bản vẽ được định trước chính là bản vẽ được lựa chọn trong danh sách 1.
8. Tách dự án nhúng ra khỏi bản vẽ được lựa chọn trong danh sách 1 (chỉ có hiệu lực khi bản vẽ có chứa dự án nhúng). Khi chọn nút này, người dùng sẽ được yêu cầu lưu dự án được tách ra thành một dự án độc lập. Nếu không lưu, dự án sẽ được xóa khỏi tệp bản vẽ.
9. Hiện thị trình quản lý Macro trong AutoCAD.
10. Hiện thị VBAIDE, là nơi sẽ thực hiện quá trình thiết kế mã lệnh và giao diện của chương trình.

### 2.2.1. Tạo mới, Mở và Lưu dự án VBA

### Để tạo mới dự án VBA

1. Mở cửa sổ **VBA Manager**.
2. Chọn nút lệnh **New**.

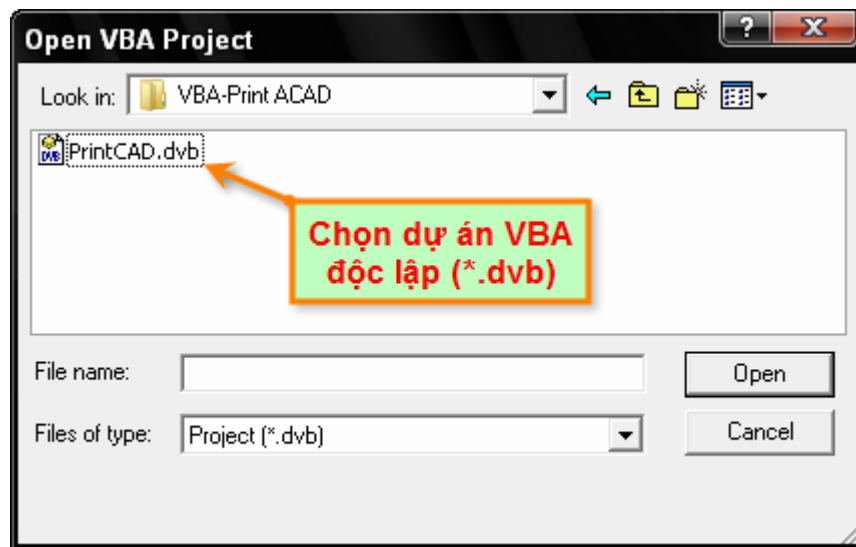
Mặc định, dự án mới được tạo sẽ là một dự án độc lập, có tên là ACADProject.

Sau khi tạo mới dự án, ta có thể nhúng dự án vào một bản vẽ nào đó hoặc có thể lưu ra một tệp riêng thành dự án độc lập tùy thuộc vào mục đích sử dụng.

Ngoài ra người dùng có thể sử dụng lệnh `VBANEW` từ dòng lệnh của AutoCAD để tạo mới dự án VBA.

### Để mở/tải dự án VBA

1. Mở cửa sổ **VBA Manager**.
2. Chọn nút lệnh **Load** ⇒ Hiện thị hộp thoại mở dự án VBA



Hình V-3: Hộp thoại mở dự án VBA.

3. Chọn dự án cần mở và chọn **Open**.

Ngoài ra người dùng có thể sử dụng lệnh `VBALOAD` từ dòng lệnh của AutoCAD để tải dự án VBA.

### Để lưu dự án VBA

1. Khởi động **VBAIDE** (nhấn phím **ALT+F11** hoặc chọn trình đơn **Tools** ⇒ **Macro** ⇒ **Visual Basic Editor**).
2. Chọn dự án cần lưu trong cửa sổ **Project**.
3. Chọn trình đơn **File** ⇒ **Save xxx** (trong đó xxx là tên tệp chứa dự án)

#### 2.2.2. Nhúng và tách dự án VBA

##### Để nhúng dự án độc lập vào một bản vẽ

1. Mở cửa sổ **VBA Manager**.
2. Trong mục **Drawing**, chọn bản vẽ cần nhúng dự án vào.
3. Trong mục **Projects**, chọn dự án cần nhúng vào bản vẽ.

## 4. Chọn nút lệnh **Embed**.

Sau khi người dùng nhúng dự án vào bản vẽ, VBAIDE sẽ sao chép toàn bộ dự án độc lập và nhúng vào bản vẽ AutoCAD, nghĩa là tại thời điểm có hai dự án VBA giống nhau cùng tồn tại, một là dự án độc lập và một là dự án nhúng trong bản vẽ.

### Để tách dự án nhúng khỏi một bản vẽ

1. Mở cửa sổ **VBA Manager**.
2. Trong mục **Drawing**, chọn bản vẽ có chứa dự án cần tách.
3. Chọn nút lệnh **Extract**.
4. Nếu cần lưu lại dự án, chọn nút lệnh **Yes** ở hộp thoại xuất hiện sau đó, nếu không có thể chọn **No**.

## 2.3. Quản lý dự án VBA từ dòng lệnh

AutoCAD có cung cấp một số câu lệnh liên quan đến việc quản lý dự án VBA và để thực thi Macro trong VBA. Dưới đây là danh sách các câu lệnh có liên quan:

Lệnh	Giải thích
VBAIDE	Mở VBAIDE. Nếu chưa có dự án nào được mở, AutoCAD sẽ tự tạo một dự án lập mới, sau đó mới hiển thị VBAIDE.
VBAMAN	Hiển thị trình quản lý dự án VBA – VBA Manager, qua đó người dùng có thể thực hiện các thao tác trên dự án VBA.
VBANEW	Tạo mới dự án độc lập.
VBALOAD	Tải tệp dự án VBA (tệp *.dwb) vào trong VBAIDE. Hộp thoại mở dự án VBA sẽ được hiển thị để người dùng lựa chọn tệp dự án cần mở.
-VBALOAD	Tải tệp dự án VBA vào trong VBAIDE như lệnh VBALOAD, nhưng không hiển thị hộp thoại, người dùng phải nhập tên tệp từ dòng lệnh của AutoCAD
VBAUNLOAD	Đóng dự án VBA đang được mở trong VBAIDE. Trên dòng lệnh AutoCAD xuất hiện dấu nhắc, nhắc người dùng nhập tên tệp chứa dự án cần đóng.
VBARUN	Thực thi Macro. AutoCAD sẽ hiển thị một hộp thoại cho người dùng chọn Macro cần thực thi.
-VBARUN	Thực thi Macro từ dòng lệnh của AutoCAD. Người dùng phải nhập tên của Macro cần thực thi ngay trên dòng lệnh của AutoCAD. Nếu có nhiều Macro trùng tên trong các mô-đun khác nhau, sử dụng cú pháp: <tên_mô_đun.tên_Macro>.
VBASTMT	Thực thi một biểu thức/câu lệnh của VBA từ dòng lệnh của AutoCAD.

Khi kết hợp cách quản lý dự án VBA bằng dòng lệnh của AutoCAD với ngôn ngữ lập trình AutoLISP thì ta có thể gọi một dự án VBA hay sử dụng một chức năng của dự án VBA bằng một chương trình AutoLISP.

## 3. Macro

### 3.1. Khái niệm Macro trong AutoCAD

Cũng tương tự như trong Excel, khái niệm về Macro cũng được đưa vào trong AutoCAD như là một công cụ giúp cho người dùng có thể thực hiện nhanh hơn công việc của mình nhờ khả năng tự động thực hiện của AutoCAD thông qua Macro, mà thực chất là một chương trình VBA. Trong AutoCAD, để được là Macro thì chương trình này phải thỏa mãn đồng thời các quy định sau:

- ✦ Là một chương trình con dạng **sub**;

- ◆ Có phạm vi là **Public**;
- ◆ Đặt trong mô-đun **chuẩn** hoặc mô-đun **ThisDrawing**.

Lưu ý rằng tên của Macro cũng chính là tên của chương trình con này.

Trong Excel, các tính năng của ứng dụng mở rộng có thể được thể hiện ở dạng Macro (để thực thi một tác vụ nào đó) hoặc ở dạng hàm (khi muốn thực hiện tính toán, thao tác có trả về giá trị). Còn trong AutoCAD, với đặc tính sử dụng chủ yếu là các thao tác trên bản vẽ nên các tính năng của ứng dụng mở rộng được thể hiện chủ yếu thông qua Macro, hàm chỉ được sử dụng nội bộ bên trong các mô-đun của dự án VBA.

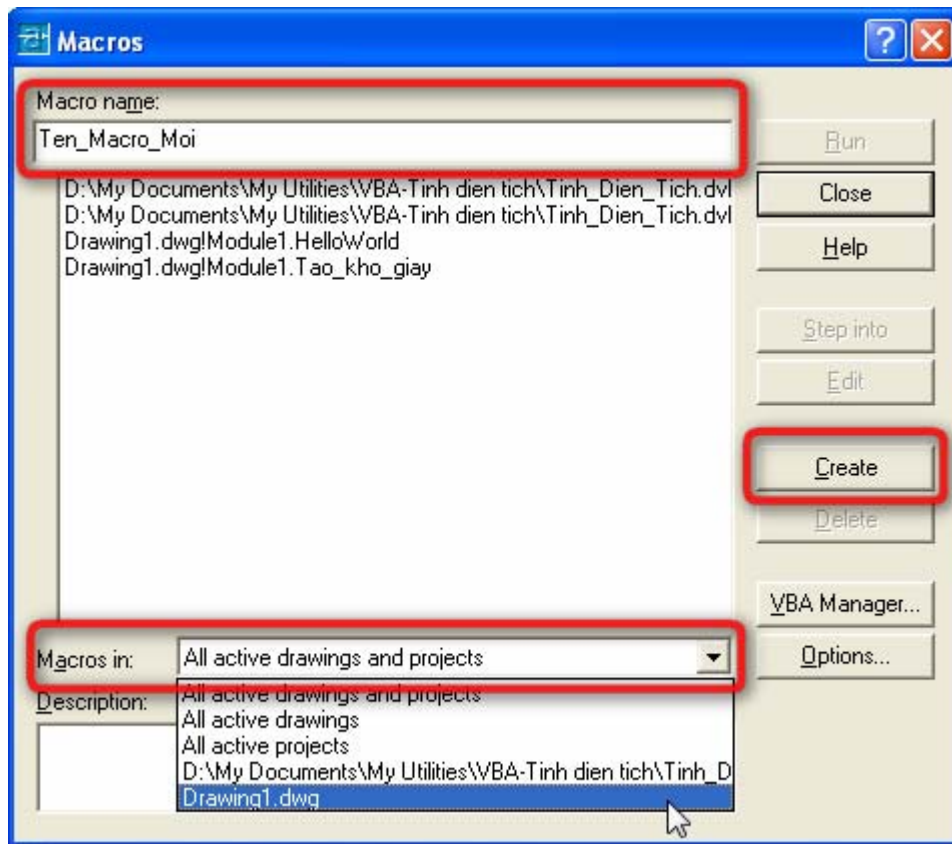
### 3.2. Tạo mới và Hiệu chỉnh Macro

Trong AutoCAD, do không có khả năng tạo Macro dạng kịch bản như trong Excel (mã lệnh của Macro được tự động sinh ra căn cứ vào sự ghi lại thao tác của người dùng trên Excel), nên để tạo mới hay hiệu chỉnh Macro trong AutoCAD, người dùng phải viết mã lệnh cho Macro trực tiếp trong VBAIDE.

Như đã đề cập, bản chất của Macro chính là một chương trình con dạng Sub trong VBA, nên để tạo mới một Macro, người dùng có thể trực tiếp vào VBAIDE và tạo ra một chương trình con thỏa mãn các quy định liên quan đến Macro trong mục **Error! Reference source not found..** AutoCAD sẽ tự động nhận diện tất cả các chương trình con phù hợp với các quy định này và xem chúng là các Macro. Tuy nhiên, người lập trình còn có thể thực hiện tạo một Macro mới thông qua giao diện hộp thoại Macros.

#### Tạo Macro thông qua hộp thoại Macros

1. Mở hộp thoại **Macros** bằng cách chọn trình đơn **Tools⇒Macro⇒Macros...** (hoặc sử dụng lệnh **VBARUN** từ dòng lệnh AutoCAD, hoặc nhấn phím tắt **ALT+F8**)



Hình V-4: Hộp thoại Macros

2. Trong mục **Macros in**, chọn nơi sẽ chứa Macro từ danh sách thả xuống.
3. Trong mục **Macro name**, nhập tên Macro cần tạo.
4. Chọn **Create** để tạo Macro mới. Nếu người dùng chưa chọn nơi chứa Macro ở bước ②, AutoCAD sẽ hiển thị hộp thoại yêu cầu người dùng chọn nơi chứa Macro.



Hình V-5: Lựa chọn nơi chứa Macro

5. Màn hình của VBAIDE sẽ được tự động hiện lên, và con trỏ chuột được đặt ngay vị trí cài đặt mã lệnh cho Macro mới. Lúc này ta đã có thể bắt đầu viết mã lệnh cho Macro mới.
6. Trong màn hình của VBAIDE chọn trình đơn **File⇒Close and Return to AutoCAD** để trở về màn hình của AutoCAD sau khi hoàn thành việc viết mã lệnh cho Macro.

**GỢI Ý** Trong AutoCAD, để hiển thị hộp thoại Macros, người lập trình có thể sử dụng phím tắt là **ALT+F8**; còn để chuyển đổi qua lại giữa màn hình AutoCAD và VBAIDE, người lập trình có thể sử dụng phím tắt là **ALT+F11**.

### Hiệu chỉnh Macro

1. Mở hộp thoại **Macros**.
2. Chọn Macro cần hiệu chỉnh trong danh sách các Macro.
3. Chọn **Edit** ⇒ AutoCAD sẽ hiển thị VBAIDE và chuyển con trỏ vào vị trí chứa mã lệnh của Macro để bắt đầu hiệu chỉnh Macro.

Tất nhiên người dùng có thể hiệu chỉnh bất cứ Macro nào mà không cần sử dụng hộp thoại **Macros**. Người dùng chỉ cần khởi động VBAIDE và tìm chương trình con tương ứng với Macro cần hiệu chỉnh, mã lệnh của chương trình con này cũng chính là mã lệnh của Macro, do đó, hiệu chỉnh chương trình con này cũng chính là hiệu chỉnh Macro.

### 3.3. Thực thi Macro

Sau khi đã viết mã lệnh cho Macro, người dùng có thể thực thi Macro trực tiếp trong VBAIDE (tham khảo mục “*Thực thi Macro trực tiếp từ VBAIDE*” trang 106). Ngoài ra, cũng có thể thực thi Macro thông qua hộp thoại Macros, hoặc thực thi Macro từ dòng lệnh AutoCAD.

#### Để thực thi Macro thông qua hộp thoại Macros

1. Mở hộp thoại **Macros**.

2. Chọn Macro cần thực thi trong danh sách các Macro.
3. Chọn **Run** ⇒ AutoCAD sẽ thực thi Macro được chọn.

### Để thực thi Macro từ dòng lệnh AutoCAD

1. Tại dòng lệnh AutoCAD, gõ lệnh `-VBARUN` ⇒ nhấn phím ENTER.
2. AutoCAD nhắc người dùng nhập tên Macro: `Macro name:` ⇒ Nhập tên Macro và nhấn phím ENTER. Nếu có nhiều Macro trùng tên nằm trong các mô-đun khác nhau, cần sử dụng cấu trúc `<Tên_mô-đun.Tên_Macro>` để chọn đúng Macro cần thực thi.

### 3.4. Định nghĩa lệnh mới bằng AutoLISP

Có thể nói, một trong những điểm mạnh của AutoCAD chính là cửa sổ dòng lệnh. Thông qua cửa sổ dòng lệnh này, mọi thao tác đối với bản vẽ đều có thể được thực hiện một cách nhanh chóng bằng bàn phím với hệ thống các lệnh đã được định nghĩa sẵn.

Để thực hiện một chức năng mới được tạo ra bằng cách lập trình trong VBA, hay nói cách khác là để thực thi một Macro, rõ ràng phải thực hiện qua khá nhiều bước. Cho nên, để tạo ra sự thuận tiện cho người sử dụng, nên định nghĩa lệnh mới, mà qua đó người sử dụng có thể thực thi Macro chỉ với một lệnh đơn giản, giống như khi cần vẽ đường thẳng, ta chỉ cần sử dụng lệnh **line** ở dòng lệnh AutoCAD. Sự kết hợp với AutoLISP là một giải pháp tốt và dễ dàng để thực hiện mục tiêu này.

Với VBA, người dùng có thể tạo Macro để thực hiện một thao tác nào đó. Còn với AutoLISP, người dùng có thể định nghĩa một lệnh mới để thực thi Macro đã được tạo trong dự án VBA. Hay nói cách khác, AutoLISP là chiếc cầu nối nhằm tạo sự liên kết giữa dòng lệnh AutoCAD và Macro trong dự án VBA.

Tuy nhiên, nếu chỉ được lưu trong những tệp dự án VBA và tệp AutoLISP thông thường thì những lệnh đó chỉ có hiệu lực khi người dùng tải đồng thời dự án VBA và tệp AutoLISP vào trong AutoCAD. Vì vậy, khi muốn các lệnh mới này có hiệu lực ngay khi sử dụng AutoCAD (nghĩa là người sử dụng không cần phải làm thêm bất cứ một thao tác nào khác, chỉ cần khởi động AutoCAD là có thể dùng được ngay các lệnh này) thì người lập trình sẽ phải lưu dự án VBA thành tệp có tên là **ACAD.DVB** và tệp AutoLISP sẽ được lưu với tên là **ACAD.LSP**, và cả hai tệp này phải được lưu vào thư mục cài đặt của AutoCAD (ví dụ đối với AutoCAD 2002, nếu cài đặt thông thường, thì thư mục cài đặt của AutoCAD trong Windows sẽ là: `C:\Program Files\AutoCAD 2002`). Đây là hai tệp sẽ được tự động tải lên ngay sau khi khởi động AutoCAD, và việc còn lại của người dùng là sử dụng tất cả những tính năng đã có trong các tệp này.

Dưới đây là một ví dụ đơn giản để tạo lệnh mới trong AutoCAD: chương trình (Macro) HelloWorld.

Mô tả nội dung chương trình (Macro): Macro này sẽ hiển thị hộp thoại yêu cầu người sử dụng nhập vào một thông điệp, sau đó thông điệp này sẽ được vẽ trên không gian mô hình của AutoCAD. Để thực thi Macro này, ta chỉ cần gõ lệnh `Hello` trong dòng lệnh AutoCAD. Và đây sẽ là lệnh mới trong AutoCAD, nó luôn sẵn sàng hoạt động ngay sau khi AutoCAD khởi động xong.

### Tạo lệnh mới trong AutoCAD bằng AutoLISP và VBA

#### 3.4.1. Tạo dự án mới

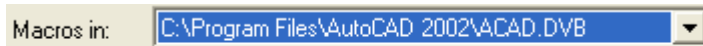
Các bước sau sẽ tạo một dự án mới và lưu trong thư mục cài đặt của AutoCAD với tên là **ACAD.DVB**.

1. Khởi động AutoCAD.

2. Mở cửa sổ **VBA Manager** (lệnh VBAMAN).
3. Chọn **New**.
4. Chọn dự án VBA vừa được tạo trong danh sách các dự án, sau đó chọn **Save As...**
5. Trong hộp thoại **Save As**, chọn thư mục cài đặt của AutoCAD trong mục **Save in**; còn trong mục **File name** nhập vào **ACAD.DVB**.
6. Chọn **Save** để lưu dự án và quay về cửa sổ **VBA Manager**.

### 3.4.2. Tạo và thử nghiệm Macro HelloWorld

7. Trong cửa sổ **VBA Manager**, chọn **Macros...** để hiển thị hộp thoại **Macros**.
8. Chọn dự án **ACAD.DVB** trong mục **Macros in**:



9. Trong mục **Macro name**, nhập vào tên Macro là **HelloWorld**



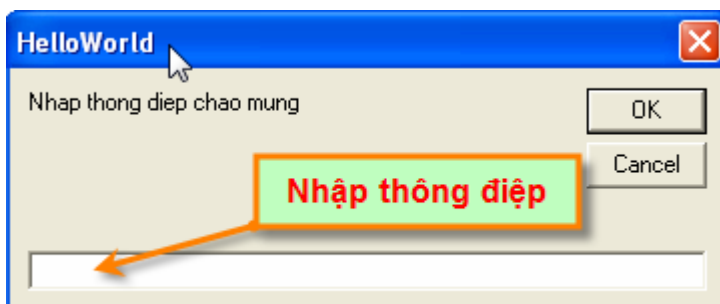
10. Chọn **Create**. Màn hình VBAIDE sẽ được hiển thị, trong cửa sổ mã lệnh, con trỏ sẽ được đặt ở vị trí của Macro vừa được tạo. Ta sẽ thấy được đoạn mã lệnh đã được tạo sẵn như sau:

```
Sub HelloWorld()  
End Sub
```

11. Thay đoạn mã lệnh trên bằng đoạn mã lệnh sau:

```
Sub HelloWorld()  
    Dim strMsg As String  
    strMsg = InputBox("Nhập thông điệp chào mừng", "HelloWorld")  
    Dim objText As AcadText  
    Dim pInsert(0 To 2) As Double  
    pInsert(0) = 50: pInsert(1) = 100: pInsert(2) = 0  
    Set objText = ThisDrawing.ModelSpace.AddText(strMsg, pInsert, 2.5)  
    ZoomExtents  
End Sub
```

12. Đặt con trỏ vào giữa hai dòng `Sub HelloWorld()` và `End Sub`, sau đó nhấn phím **F5** để thực thi thử Macro. Một hộp thoại nhỏ sẽ hiện lên yêu cầu người dùng nhập vào một thông điệp ⇒ Nhập vào thông điệp và nhấn **OK** ⇒ Thông điệp vừa nhập sẽ được vẽ trên không gian mô hình của AutoCAD.





13. Trong màn hình của VBAIDE, chọn trình đơn **File** ⇒ **Save** để lưu tệp dự án **ACAD.DVB**. Tiếp tục chọn trình đơn **File** ⇒ **Close and Return to AutoCAD** để trở về AutoCAD.

### 3.4.3. Tạo lệnh mới bằng AutoLISP

Các bước tiếp sau sẽ tiến hành tạo một tệp **ACAD.LSP** mới, khai báo một lệnh mới trong AutoCAD thông qua AutoLISP và lệnh đó sẽ thực thi Macro HelloWorld vừa được tạo.

**CHÚ Ý** Nếu trong thư mục cài đặt của AutoCAD đã có tệp **ACAD.LSP**, chỉ cần sao chép đoạn mã ở bước 15 và thêm vào cuối tệp **ACAD.LSP** đã có.

14. Khởi động chương trình soạn thảo văn bản, ví dụ như chương trình **Notepad** có sẵn trong Windows. Trong Windows, chọn trình đơn **Start** ⇒ **Run**. Trong hộp thoại **Run**, nhập **notepad** sau đó nhấn **OK** để khởi động trình soạn thảo văn bản **Notepad**.
15. Trong chương trình **Notepad**, nhập vào đoạn khai báo sau:

```
(defun C:Hello ()
  (command "-vbarun" "HelloWorld")
)
```

Trong đó **Hello** là lệnh được khai báo để đăng ký sử dụng trong AutoCAD. Còn **HelloWorld** là tên Macro đã tạo ra trong VBA ở các bước trước.

16. Chọn trình đơn **File** ⇒ **Save**. Trong hộp thoại **Save As...**, chọn thư mục cài đặt của AutoCAD trong mục **Save in**, trong mục **File name** nhập vào **ACAD.LSP** ⇒ Chọn **Save**.
17. Thoát khỏi chương trình AutoCAD (nếu đang thao tác trên AutoCAD). Khởi động lại chương trình AutoCAD. Trong dòng lệnh của AutoCAD, gõ lệnh **Hello**. Thật bất ngờ, một hộp thoại yêu cầu người dùng nhập thông điệp chào mừng xuất hiện - Macro **HelloWorld** đã được thực thi.

Như vậy, bằng cách sử dụng VBA kết hợp với AutoLISP, người lập trình có thể tạo thêm rất nhiều lệnh mới trong AutoCAD một cách dễ dàng.

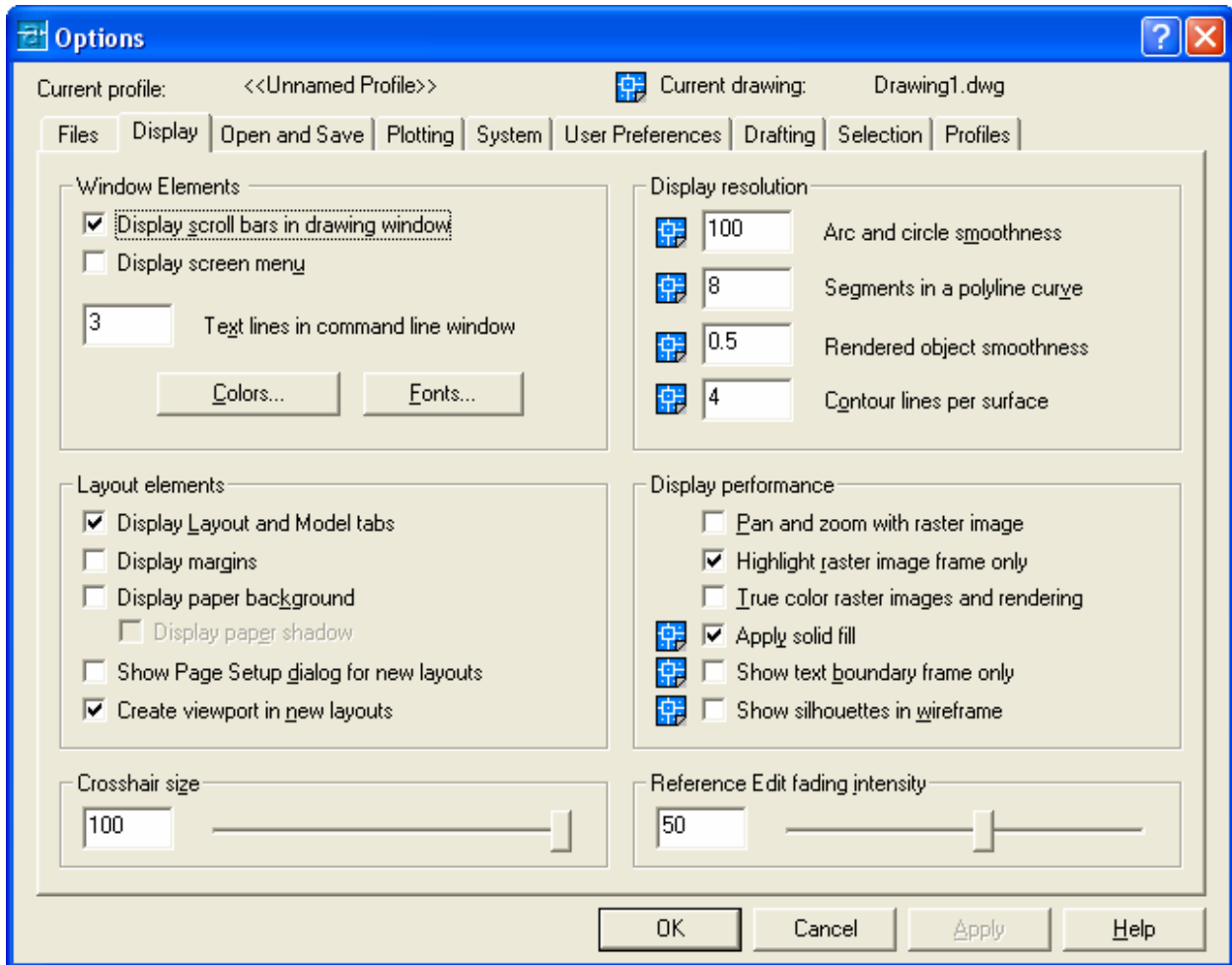
Để hiểu hơn về AutoLISP, có thể đọc thêm tài liệu hướng dẫn có sẵn trong AutoCAD.

## 4. Hệ thống đối tượng trong AutoCAD

### 4.1. Mô hình đối tượng trong AutoCAD

AutoCAD được tạo thành từ nhiều thành phần khác nhau và chúng luôn có một mối liên hệ chặt chẽ được quy định từ trước theo một cấu trúc nhất định. Nếu chỉ sử dụng AutoCAD với các lệnh có sẵn để vẽ (mức độ thông thường) thì người dùng không cần biết đến cấu trúc này, đối với họ, khả năng và mức độ tiện dụng của các lệnh này mới là quan trọng. Nhưng khi sử dụng AutoCAD như là một ứng dụng nền để lập trình, thì lúc này có nhiều khái niệm phải được hiểu theo cách khác, thiết thực cho việc lập trình. AutoCAD được coi như là một đối tượng được cấu thành từ những đối tượng khác, những đối tượng con này, theo cách hiểu thông thường, chính là các thành phần của AutoCAD. Trong từng đối tượng con đó, có thể có nhiều đối tượng ở cấp độ thấp hơn nữa, hay nói cách khác, một đối tượng có thể bao gồm nhiều đối tượng con với các cấp khác nhau. Các đối tượng này được tổ chức chặt chẽ theo một cấu trúc cố định, và dựa vào cấu trúc tổ chức đối tượng này, AutoCAD cho người dùng cái nhìn tổng quan về cấu tạo của nó cũng như chỉ cho người dùng biết cần phải làm như thế nào để tác động lên một đối tượng cụ thể (hay thành phần) của nó. Thông thường cấu trúc này được gọi là **mô hình đối tượng** trong AutoCAD và được biểu diễn dưới dạng cấu trúc cây phân nhánh. Với cấu trúc này, người dùng có thể dễ dàng truy cập đến đối tượng mình cần thao tác bằng cách tra cứu theo nhánh của đối tượng đó.

Mỗi đối tượng trong cấu trúc trên sẽ tương đương với một thành phần của AutoCAD, ví dụ như đối tượng Preferences sẽ tương đương với hộp thoại sau trong AutoCAD:



Những thao tác bằng mã lệnh tác động lên đối tượng Preferences sẽ tương đương với việc người dùng thao tác trực tiếp lên hộp thoại Options từ trong AutoCAD.

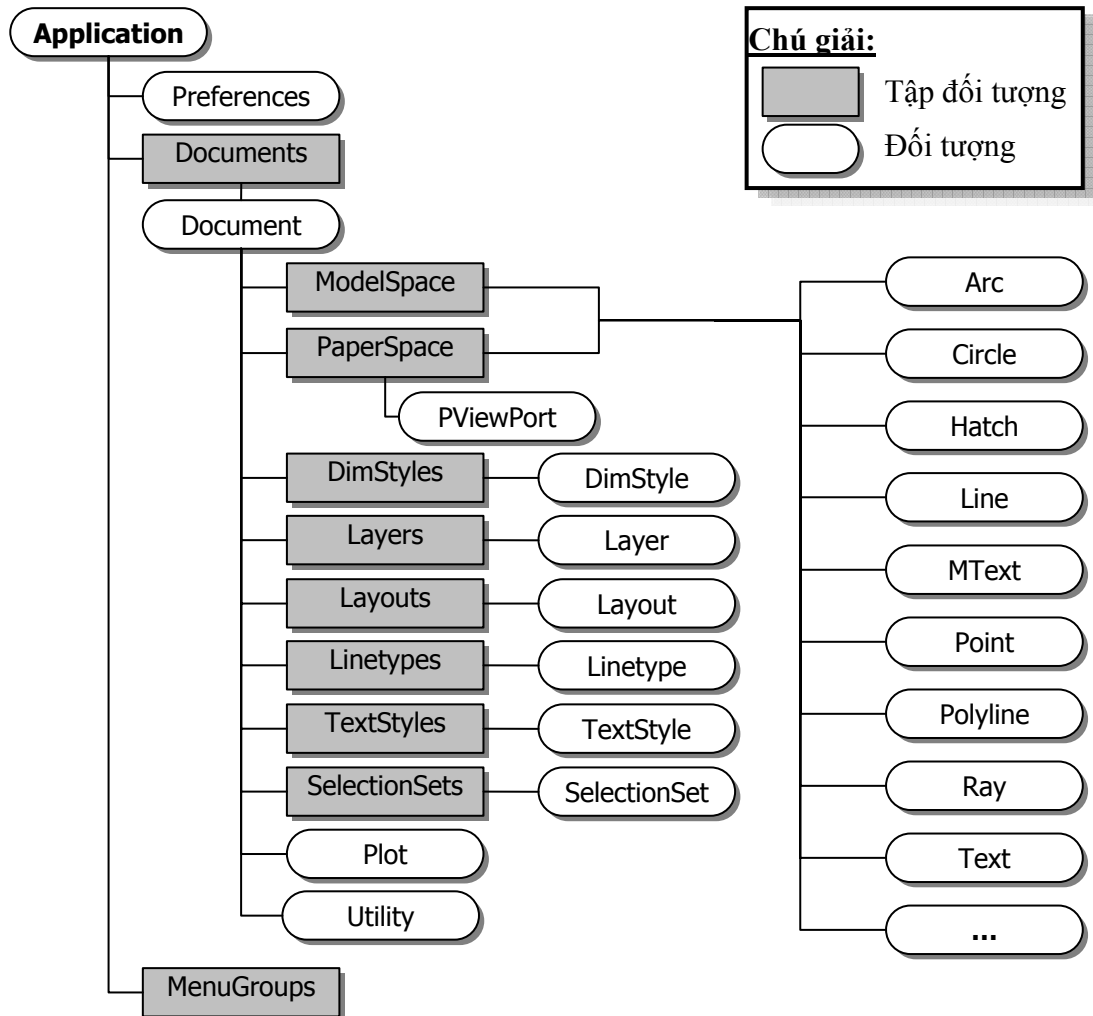
Như vậy có thể thấy rằng, để lập trình tốt trên AutoCAD, người dùng, trước hết, phải quen thuộc và thao tác thành thạo trên các thành phần (hay đối tượng) của AutoCAD.

Trong AutoCAD, có rất nhiều loại đối tượng khác nhau, chẳng hạn như:

- ◆ Các đối tượng hình học: line, arc, text, dimension...
- ◆ Thiết lập về định dạng: linetype, dimension style...
- ◆ Cấu trúc tổ chức: layer, group, block...
- ◆ Đối tượng liên quan đến hiển thị bản vẽ: view, viewport,...
- ◆ Và ngay cả bản vẽ và bản thân chương trình AutoCAD cũng được xem là đối tượng.

Mô hình đối tượng của AutoCAD trong VBA được thể hiện dưới dạng cấu trúc cây phân cấp, trong đó đối tượng gốc là Application, chính là phiên bản AutoCAD đang chạy. Nhờ có mô hình đối tượng mà người lập trình có thể biết được một đối tượng có thể cho phép truy cập đến những đối tượng nào ở cấp tiếp theo.

Dưới đây là mô hình đối tượng rút gọn trong AutoCAD dùng cho việc lập trình bằng VBA. Mô hình đối tượng đầy đủ có thể tham khảo trong tài liệu “ActiveX and VBA Developer's Guide” đi kèm AutoCAD.



Hình V-6: Mô hình đối tượng trong AutoCAD.

Mỗi đối tượng (Object), cũng giống như một vật thể, đều có những tính chất và những hành vi đặc trưng cho nó. Trong lập trình, tính chất của đối tượng được biểu diễn thông qua khái niệm **thuộc tính**, còn hành vi được biểu diễn thông qua khái niệm **phương thức**. Chẳng hạn như đối tượng `Application`, là đối tượng thể hiện cho chương trình AutoCAD, có thuộc tính `Caption` chứa tiêu đề của chương trình AutoCAD và phương thức `Quit` dùng để thoát khỏi chương trình AutoCAD. Để truy cập đến các thành phần (phương thức, thuộc tính, ...) của đối tượng, ta sử dụng quy tắc dấu chấm (.):

`<Tên đối tượng>.<Tên phương thức/Thuộc tính>`

Các đối tượng có những điểm chung nhau còn có thể được nhóm lại và được biểu diễn thông qua **tập đối tượng** (collection). Mỗi một tập đối tượng có các phương thức và thuộc tính riêng để người dùng tác động lên nó như: thêm đối tượng tập đối tượng bằng phương thức `Add` (đúng với hầu hết các loại tập đối tượng), thuộc tính `Count` dùng để đếm số đối tượng trong tập đối tượng, phương thức `Item` sử dụng để truy cập bất kỳ đối tượng nào trong tập đối tượng.

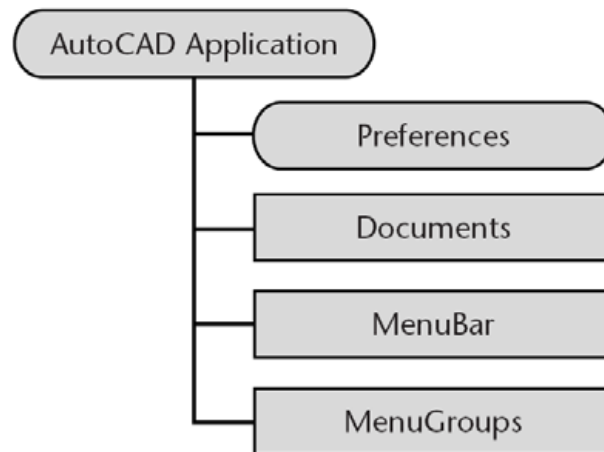
## 4.2. Một số đối tượng chính trong AutoCAD

### 4.2.1. Đối tượng Application

Đối tượng `Application` là đối tượng thể hiện cho một phiên làm việc của AutoCAD, đối tượng này sẽ được tự động tạo ra mỗi khi khởi động chương trình AutoCAD. Tất cả các thành phần và thao tác thực hiện trong cửa sổ chính của chương trình AutoCAD đều được thể hiện thông qua các phương thức và thuộc tính của đối tượng `Application`. Ví dụ, đối tượng `Application` có thuộc tính `Preferences` trả về đối tượng `Preferences`. Đối tượng này cho phép truy cập đến các

cấu hình bên trong của hộp thoại Option. Các thuộc tính khác của đối tượng Application cho phép truy cập đến các dữ liệu riêng của chương trình chẳng hạn như tên và phiên bản chương trình, kích thước, vị trí của cửa sổ ... . Các phương thức của đối tượng Application sẽ thực hiện các thao tác như: tạo mới, mở, đóng bản vẽ hay thoát khỏi AutoCAD.

Đối tượng Application là đối tượng gốc trong mô hình đối tượng của AutoCAD. Từ đối tượng Application, ta có thể truy xuất đến bất kỳ đối tượng nào, chẳng hạn như đối tượng Application có các liên kết đến bản vẽ AutoCAD thông qua tập đối tượng Documents, các trình đơn và thanh công cụ AutoCAD thông qua tập đối tượng MenuBar và MenuGroups, và VBAIDE thông qua một thuộc tính gọi là VBE.

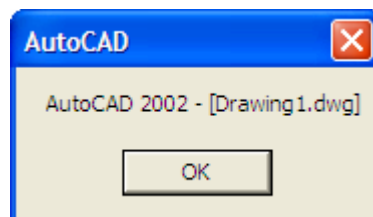


**Hình V-7: Các thành phần của đối tượng Application.**

Đối tượng Application là đối tượng toàn cục. Điều này có nghĩa là tất cả các phương thức và thuộc tính của đối tượng Application luôn có hiệu lực trong VBAIDE, tức là khi truy cập đến các phương thức và thuộc tính của đối tượng Application đều không cần có tiền tố Application ở trước nên hai câu mã lệnh dưới đây đều có tác dụng như nhau là thông báo nội dung thanh tiêu đề của ứng dụng AutoCAD đang chạy:

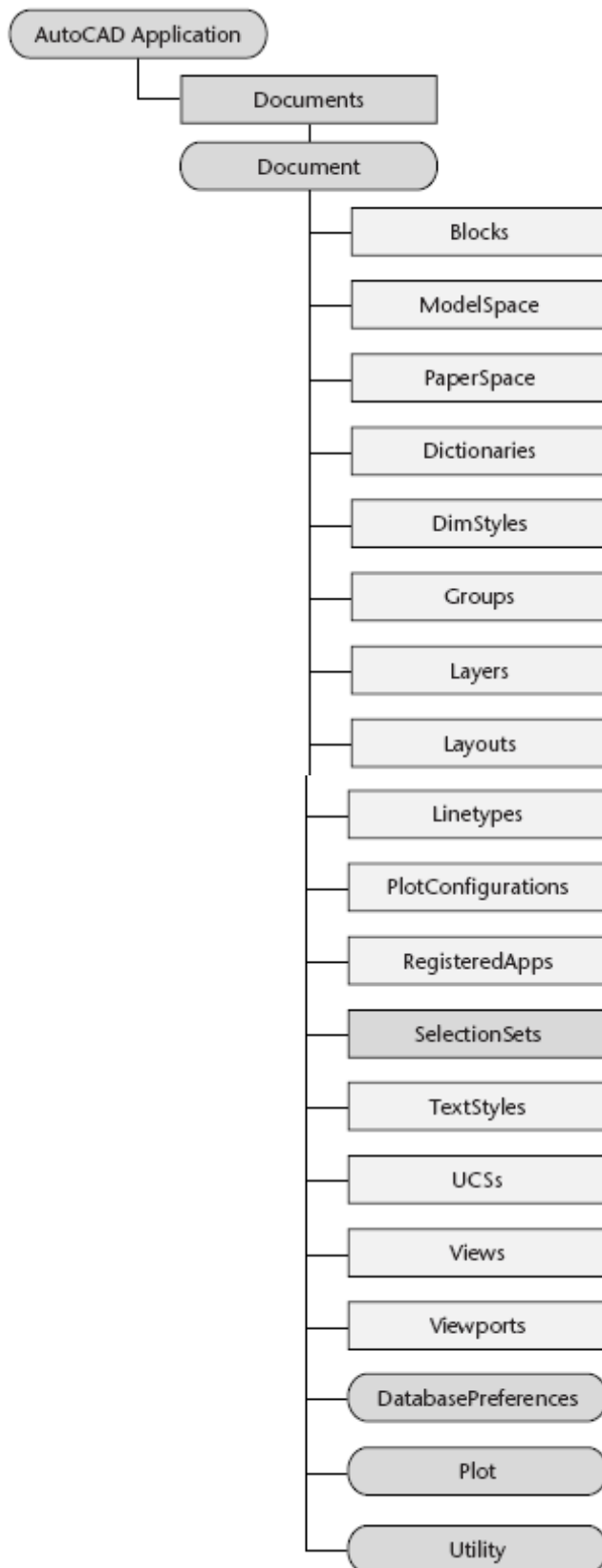
```
MsgBox Application.Caption  
MsgBox Caption
```

Thông báo có thể xuất hiện như hình dưới:



### 4.2.2. Đối tượng Document

Đối tượng Document, thực chất là một bản vẽ AutoCAD đang được mở, thuộc tập đối tượng Documents (tương đương với tất cả các bản vẽ đang được mở), cho nên nó chứa tất cả các đối tượng hình học và phi hình học trong một bản vẽ AutoCAD cũng như chứa hầu hết các đối tượng (hay thành phần) khác của bản vẽ như Views hay Viewports. Để truy cập đến các đối tượng của một bản vẽ ta cần phải thông qua đối tượng Document tương ứng với bản vẽ đó. Như trong mô hình đối tượng ở trên, các đối tượng hình học (đường thẳng, hình tròn, cung, ...) được truy cập thông qua tập đối tượng ModelSpace và PaperSpace, còn các đối tượng phi hình học (layer, linetype, text style, ...) được truy cập thông qua tập đối tượng có tên tương ứng, chẳng hạn như Layers, Linetypes, TextStyles.



Trong mỗi dự án VBA, ThisDrawing là một đối tượng kiểu Document và luôn có sẵn. Với đối tượng ThisDrawing này, người dùng không cần phải khai báo hoặc gán giá trị cho đối tượng này mà có thể truy cập được ngay do nó luôn tồn tại trong dự án VBA. ThisDrawing tham chiếu đến bản vẽ hiện hành trong AutoCAD, nghĩa là những tác động lên đối tượng này sẽ tương đương với việc tác động lên bản vẽ hiện hành trong AutoCAD. Một đối tượng tương

đương với ThisDrawing chính là đối tượng ActiveDocument, đối tượng này cũng là một thuộc tính của đối tượng Application.

Để hiển thị tên bản vẽ hiện hành trong AutoCAD ta có thể truy cập theo những cách sau (từ mô-đun ThisDrawing của dự án VBA):

```
MsgBox Name  
MsgBox ThisDrawing.Name  
MsgBox Application.ActiveDocument.Name
```

### 4.2.3. Tập đối tượng

AutoCAD tổ chức hầu hết các đối tượng vào trong tập đối tượng, ví dụ như tất cả các đối tượng hình học, cho dù khác nhau về loại đối tượng, đều được đặt trong tập đối tượng ModelSpace, PaperSpace và Block. Để truy cập vào một đối tượng nào đó ta phải thông qua tập đối tượng chứa nó.

Mỗi một tập đối tượng có một phương thức dùng để thêm đối tượng vào bản thân tập đối tượng đó và hầu hết các tập đối tượng đều sử dụng phương thức Add để thực hiện nhiệm vụ này. Chú ý rằng, trong AutoCAD, khi thêm các đối tượng hình học vào tập đối tượng liên quan (như ModelSpace và PaperSpace) thì phương thức dùng để thực hiện nhiệm vụ này có tên là Add<Tên đối tượng>, ví dụ để thêm vào một đường thẳng (Line) ta sử dụng phương thức AddLine. Trong khi đó, đối với các đối tượng khác, ví dụ như các đối tượng phi hình học như Layer chẳng hạn, thì phương thức của tập đối tượng Layers dùng để thêm một đối tượng vào trong tập đối tượng lại có tên là Add.

Cách thức thêm đối tượng vào tập đối tượng:

```
ModelSpace.AddLine (P1, P2)  
Layers.Add ("ABC")
```

Các tập đối tượng có những phương thức và thuộc tính giống nhau dùng để thao tác với chúng cũng như với các đối tượng bên trong chúng. Ví dụ thuộc tính Count dùng để truy cập bộ đếm số đối tượng có trong tập đối tượng. Phương thức Item sử dụng để truy cập bất kỳ đối tượng nào trong tập đối tượng.

Đoạn mã sau sẽ hiển thị số đối tượng hình học hiện có trong bản vẽ và tên của đối tượng hình học đầu tiên:

```
MsgBox ModelSpace.Count  
MsgBox ModelSpace.Item(0).ObjectName
```

### 4.2.4. Đối tượng phi hình học

Các đối tượng phi hình học là những đối tượng không thể nhìn thấy được, chúng được sử dụng trong AutoCAD để thiết lập các thuộc tính cho đối tượng hình học. Những đối tượng phi hình học hay gặp là: Layer, Linetype, DimStyle, ... Các đối tượng phi hình học thường được chứa trong các tập đối tượng có tên tương ứng, ví dụ như Layers, Linetypes, DimStyles, ...

Cách thức để tạo ra một đối tượng phi hình học là sử dụng phương thức Add của đối tượng tập đối tượng tương ứng. Ví dụ sau sẽ tạo ra một Layer mới có tên là "ABC":

```
Layers.Add ("ABC")
```

Để hiệu chỉnh và truy vấn các đối tượng phi hình học, sử dụng các phương thức và thuộc tính riêng trong từng đối tượng tương ứng. Ví dụ sau sẽ thay đổi màu của Layer "ABC" thành màu đỏ:

```
Layers("ABC").Color = acRed
```

Mỗi loại đối tượng phi đồ họa đều có các phương thức để thiết lập và gọi lại dữ liệu mở rộng (xdata) và xoá bản thân đối tượng. Ví dụ sau sẽ xoá lớp “ABC”:

```
Layers("ABC").Delete
```

Cách thức thao tác trên các đối tượng phi hình học sẽ được trình bày cụ thể ở phần “*Các thao tác cơ bản trong AutoCAD*” trang 200.

#### 4.2.5. Đối tượng hình học

Đối tượng hình học hay còn gọi là thực thể, là những đối tượng hữu hình cấu thành bản vẽ của AutoCAD, một số đối tượng điển hình loại này là: đường thẳng (Line), hình tròn (Circle), .... Để tạo những đối tượng này, ta sử dụng phương thức Add<Tên thực thể> của tập đối tượng tương ứng. Để hiệu chỉnh hoặc truy vấn các đối tượng, ta sử dụng các phương thức và thuộc tính của bản thân từng đối tượng.

Mỗi đối tượng hình học đều có các thuộc tính cho phép hiệu chỉnh đối tượng như Copy, Erase, Move, Mirror... . Lưu ý rằng, những thuộc tính này sẽ tác động lên đối tượng tương tự như khi ta sử dụng các lệnh tương ứng trong AutoCAD để hiệu chỉnh đối tượng.

Những đối tượng hình học còn có các phương thức để xác lập và gọi lại các dữ liệu mở rộng (xdata), lựa chọn và cập nhật, lấy hình bao của đối tượng. Trong các đối tượng hình học đều có các thuộc tính điển hình như Layer, Linetype, Color, và Handle cũng như những thuộc tính riêng biệt, phụ thuộc vào loại đối tượng, chẳng hạn như Center, Radius, và Area.

Dưới đây là các phương thức và thuộc tính có trong hầu hết các đối tượng hình học.

#### Các phương thức của đối tượng hình học

Phương thức	Giải thích
ArrayPolar	Nhân bản dạng cực đối tượng được chọn (giống như lệnh <code>array</code> ) dựa trên số đối tượng cần nhân bản, góc quay cần để tạo đối tượng và tâm của cung tròn.
ArrayRectangular	Nhân bản dạng chữ nhật đối tượng được chọn (giống như lệnh <code>array</code> ) dựa trên số hàng, số cột, số tầng và các khoảng các tương ứng.
Copy	Sao chép đối tượng được chọn. Đối tượng mới được tạo sẽ có vị trí trùng với đối tượng gốc.
GetBoundingBox	Phương thức này trả về toạ độ hai điểm cấu thành hình chữ nhật bao đối tượng được chọn.
Highlight	Định trạng thái của đối tượng: có đang được chọn hay không.
IntersectWith	Trả về toạ độ các điểm mà đối tượng được chọn giao với các đối tượng khác. Người lập trình cũng có thể thiết lập các chế độ khác nhau trong quá trình tìm giao với các đối tượng khác.
Mirror	Lấy đối xứng đối tượng qua một đường thẳng đi qua hai điểm do người dùng định ra.
Move	Di chuyển đối tượng được chọn theo vector xác định bằng hai điểm do người dùng định ra.
Rotate	Xoay đối tượng quanh một điểm.
ScaleEntity	Co giãn đối tượng được chọn theo một tỉ lệ nhất định với một điểm cơ sở cho trước.
Update	Cập nhật đối tượng trên màn hình bản vẽ.

**Các thuộc tính của đối tượng hình học**

<b>Thuộc tính</b>	<b>Giải thích</b>
Color	Xác định màu cho đối tượng. Giá trị màu có thể nhập là số nguyên từ 0 đến 256 hoặc theo các hằng số đã được định nghĩa trước trong VBA.
Layer	Xác định lớp cho đối tượng được chọn.
Linetype	Xác định kiểu đường cho đối tượng được chọn.
LinetypeScale	Xác định tỉ lệ kiểu đường cho đối tượng được chọn.
Lineweight	Xác định bề dày nét vẽ của đối tượng được chọn.
Visible	Xác định tính nhìn thấy của đối tượng trong bản vẽ.

**5. Các thao tác cơ bản trong AutoCAD**

**5.1. Điều khiển AutoCAD**

**5.1.1. Tạo mới, Mở, Lưu và Đóng bản vẽ**

Những thao tác này được thực hiện thông qua việc truy cập đến tập đối tượng Documents và đối tượng Document.

Để tạo một bản vẽ mới, hoặc mở một bản vẽ đã có, ta phải sử dụng các phương thức trong tập đối tượng Documents. Phương thức Add sẽ tạo một bản vẽ mới và thêm bản vẽ đó vào tập đối tượng Documents. Phương thức Open sẽ mở một bản vẽ đã có và cũng sẽ thêm bản vẽ đó vào tập đối tượng Documents. Ngoài ra còn có phương thức Close trong tập đối tượng Documents dùng để đóng tất cả các bản vẽ đang mở trong phiên làm việc của AutoCAD.

Để lưu, nhập hoặc xuất một bản vẽ, ta sử dụng các phương thức của đối tượng Document: Save, Save As, Import và Export.

**Mở bản vẽ**

Để mở bản vẽ, sử dụng phương thức Open có trong tập đối tượng Documents. Bản vẽ vừa được mở sẽ được chuyển thành bản vẽ hiện hành. Cú pháp của phương thức Open như sau:

```
object.Open Name[, ReadOnly]
```

Object ở đây là tập đối tượng Documents hoặc một đối tượng có kiểu là Document. Ý nghĩa của các tham số như sau:

<b>Tham số</b>	<b>Giải thích</b>
Name	Là chuỗi ký tự chứa đường dẫn đầy đủ đến tập bản vẽ cần mở.
ReadOnly	Là tham số tùy chọn. Nếu gán tham số này bằng TRUE, bản vẽ được mở ra với thuộc tính chỉ đọc, nghĩa là người dùng không thể lưu bản vẽ. Giá trị mặc định của thuộc tính này là FALSE.

Ví dụ sau sử dụng phương thức Open để mở một bản vẽ đã có. Khi thực hành, cần thay đổi tên bản vẽ hoặc đường dẫn cho biến dwgName để chỉ đến một bản vẽ hiện có trong máy tính.

```
Sub OpenDrawing()
    Dim dwgName As String
    dwgName = "C:\Program Files\AutoCAD 2002\Sample\campus.dwg"
    On Error Resume Next
    Application.Documents.Open dwgName           \←Mở bản vẽ
```



```
If Err.Description <> "" Then                                     \←Thông báo khi có lỗi
    MsgBox "File " & dwgName & " does not exist."
    Err.Clear
End If
End Sub
```

### Tạo bản vẽ mới

Để tạo bản vẽ mới, sử dụng phương thức Add có trong tập đối tượng Documents. Giá trị trả về của phương thức này là một đối tượng kiểu Document chứa bản vẽ vừa được tạo. Cú pháp của phương thức Open như sau:

```
SetRetVal = Documents.Add([TemplateName])
```

Tham số	Giải thích
TemplateName	Tham số tùy chọn. Là chuỗi ký tự chứa đường dẫn đầy đủ đến tệp bản vẽ mẫu (tệp *.dwt). Nếu không nhập tham số này, AutoCAD sẽ tạo bản vẽ dựa trên tệp bản vẽ mẫu mặc định (thường có tên là Acad.dwt).
RetVal	Đối tượng kiểu Document chứa bản vẽ vừa mới tạo.

Ví dụ sau sử dụng phương thức Add để tạo một bản vẽ mới dựa trên tệp bản vẽ mẫu mặc định.

```
Sub NewDrawing()
    Dim docObj As AcadDocument
    Set docObj =Documents.Add                                     \← Tạo bản vẽ mới
End Sub
```

### Lưu bản vẽ

Để lưu bản vẽ, có thể sử dụng phương thức Save (lưu bản vẽ với tên hiện hành) hoặc SaveAs (lưu bản vẽ với tên khác). Cú pháp của các phương thức trên như sau:

```
Object.Save
Object.SaveAs FileName[, FileType]
```

Tham số	Giải thích
Object	Đối tượng kiểu Document, là bản vẽ sẽ được lưu.
FileName	Là chuỗi ký tự chứa tên tệp sẽ được lưu (bao gồm cả đường dẫn đầy đủ). Nếu không chỉ ra đường dẫn đầy đủ, tệp bản vẽ sẽ được lưu vào thư mục hoạt động của AutoCAD (thông thường là C:\Program Files\AutoCAD 2002).

FileType	Tham số tùy chọn, là hằng số xác định kiểu tệp bản vẽ sẽ được lưu. Các hằng số có thể là:
acR14_DWG	: tệp AutoCAD Release14/LT97 DWG (*.dwg)
acR14_DXF	: tệp AutoCAD Release14/LT97 DXF (*.dxf)
acR15_DWG	: tệp AutoCAD 2000 DWG (*.dwg)
acR15_DXF	: tệp AutoCAD 2000 DXF (*.dxf)
acR15_Template	: tệp AutoCAD 2000 Drawing Template File (*.dwt)
acNative	: tệp bản vẽ được lưu với kiểu tệp mới nhất ứng với phiên bản AutoCAD hiện hành. Trong AutoCAD 2002, giá trị này tương đương với hằng số acR15_DWG.

Ví dụ sau sẽ lưu bản vẽ hiện hành sử dụng tên tệp sẵn có đồng thời cũng lưu bản vẽ với một tên khác.

```
Sub SaveActiveDrawing()
    ' Lưu bản vẽ hiện hành sử dụng tên tệp sẵn có
    ThisDrawing.Save
    ' Lưu bản vẽ sử dụng tên khác
    ThisDrawing.SaveAs "MyDrawing.dwg"
End Sub
```

Thông thường, trước khi thoát khỏi phiên làm việc của AutoCAD hoặc trước khi đóng bản vẽ, người lập trình thường muốn kiểm tra xem bản vẽ đã được lưu đổi hay chưa. Để làm được việc này, có thể sử dụng thuộc tính Saved có trong đối tượng chứa bản vẽ đó (đối tượng Document tương ứng).

Ví dụ sau sẽ kiểm tra xem bản vẽ đã được lưu hay chưa và sẽ hỏi người dùng xem có đồng ý để lưu bản vẽ hay không, nếu không đồng ý, sẽ thoát khỏi chương trình. Nếu đồng ý, sẽ sử dụng phương thức Save để lưu bản vẽ hiện hành.

```
Sub TestIfSaved()
    If Not (ThisDrawing.Saved) Then
        If MsgBox("Do you wish to save this drawing?", _
            vbYesNo) = vbYes Then
            ThisDrawing.Save
        End If
    End If
End Sub
```

## Đóng bản vẽ

Để đóng bản vẽ, sử dụng phương thức Close có trong đối tượng Document. Cú pháp của phương thức Close như sau:

```
object.Close([SaveChanges][, FileName])
```

Tham số	Giải thích
object	Đối tượng kiểu Document, là đối tượng chứa bản vẽ cần đóng.
SaveChanges	Tham số tùy chọn, xác định xem có cần phải lưu bản vẽ lại trước khi đóng hay không. Nếu bằng TRUE, sẽ lưu bản vẽ, ngược lại là FALSE. Giá trị mặc định của tham số này là TRUE.

FileName	Tham số tùy chọn, xác định tên của bản vẽ sẽ được lưu trong trường hợp bản vẽ chưa được lưu lần nào.
----------	--

Trong trường hợp chưa có sự thay đổi trong bản vẽ, các tham số trên được bỏ qua và phương thức Close chỉ đơn giản là đóng bản vẽ đang được tham chiếu. Nếu đã có sự thay đổi trong bản vẽ, tham số SaveChanges sẽ xác định xem bản vẽ có được lưu hay không:

- ◆ Nếu SaveChanges bằng TRUE và bản vẽ chưa được lưu lần nào, tham số FileName sẽ được dùng để làm tên tệp lưu bản vẽ. Nếu không có tham số FileName, bản vẽ được lưu với tên mặc định trong thư mục hiện hành của AutoCAD. Trong trường hợp bản vẽ đã được lưu trước đó, tham số FileName sẽ bị bỏ qua.
- ◆ Nếu SaveChanges bằng FALSE, bản vẽ sẽ được đóng mà không được lưu.

Ví dụ sau sẽ hỏi người dùng có muốn đóng bản vẽ hay không, sau đó kiểm tra xem tệp đã được lưu lần đầu chưa, tiếp đó mới thực sự đóng bản vẽ lại sử dụng phương thức Close có trong đối tượng bản vẽ hiện hành.

```
Sub CloseDrawing ()
    If MsgBox("Bạn có muốn đóng bản vẽ: " & ThisDrawing.WindowTitle, _
        vbYesNo + vbQuestion) = vbYes Then
        If ThisDrawing.FullName <> "" Then
            ThisDrawing.Close SaveChanges:=True '←Đóng bản vẽ hiện hành
        Else
            MsgBox(ThisDrawing.Name & " chưa được lưu nên không thể đóng!")
        End If
    End If
End Sub
```

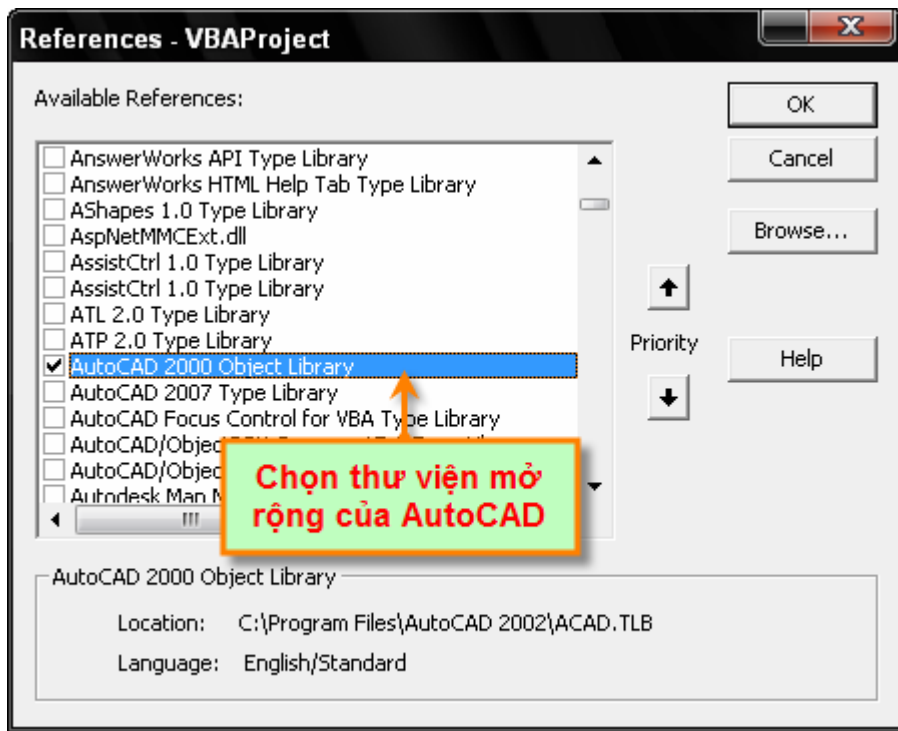
Ngoài ra, người lập trình có thể sử dụng phương thức Close có trong tập đối tượng Documents để đóng tất cả các bản vẽ hiện đang có trong phiên làm việc hiện hành của AutoCAD. Phương thức này thực hiện tương tự như khi sử dụng phương thức Close cho từng đối tượng bản vẽ với tham số SaveChanges được gán bằng TRUE. Do không kiểm soát được quá trình đóng của từng bản vẽ nên phương thức Close của tập đối tượng Documents nên hạn chế sử dụng.

### 5.1.2. Khởi động và thoát khỏi chương trình AutoCAD

#### Khởi động chương trình AutoCAD

Do được thực thi bên trong AutoCAD nên các chương trình viết bằng VBA trong AutoCAD không cần phải thực hiện thao tác khởi động chương trình AutoCAD. Tuy nhiên, khi người dùng viết mã lệnh từ các ứng dụng nền khác, chẳng hạn như viết chương trình bằng VBA trong Excel, thì cần thiết phải viết mã lệnh khởi động chương trình AutoCAD. Thực chất của đoạn mã lệnh này là tạo ra đối tượng Application.

Việc khởi động chương trình AutoCAD từ một chương trình ngoài cũng cần phải thực hiện các thao tác tương tự như khi khởi động chương trình Excel từ chương trình ngoài (tham khảo mục “*Khởi động Excel từ chương trình khác*” trang 148). Ở đây, người dùng sẽ phải tham chiếu đến thư viện mở rộng của AutoCAD với tên là “**AutoCAD 2000 Object Library**”. Với các phiên bản của chương trình AutoCAD khác nhau thì tên thư viện mở rộng có thể khác nhau.



Hình V-8: Hộp thoại References trong VBAIDE của Excel.

Đoạn mã lệnh sau sẽ khởi động chương trình AutoCAD từ VBA trong Excel.

```
Sub ConnectToAcad()
    Dim acadApp As AcadApplication
    On Error Resume Next

    Set acadApp = GetObject(, "AutoCAD.Application")
    If Err Then
        Err.Clear
        Set acadApp = CreateObject("AutoCAD.Application")
        If Err Then
            MsgBox Err.Description
            Exit Sub
        End If
    End If
    acadApp.Visible = True
    `===== Hết đoạn chương trình khởi động AutoCAD =====

    ` Hiển thị tên chương trình và phiên bản của AutoCAD
    MsgBox "Now running " + acadApp.Name + _
        " version " + acadApp.Version
End Sub
```

### Thoát khỏi chương trình AutoCAD

Việc thoát khỏi AutoCAD rất đơn giản, chỉ cần thực hiện phương thức Quit có trong đối tượng Application. Phương thức này sẽ đóng tất cả các bản vẽ và dự án VBA trong AutoCAD lại, nếu có bản vẽ hoặc dự án nào chưa được lưu, nó sẽ nhắc người dùng lưu bản vẽ, sau đó mới thực sự thoát khỏi AutoCAD.

Đoạn mã lệnh sau sẽ đóng chương trình AutoCAD.

```
Sub Thoat_AutoCAD()
    Application.Quit
```

```
End Sub
```

### 5.1.3. Sử dụng các lệnh sẵn có của AutoCAD

Các lệnh sẵn có của AutoCAD có thể được sử dụng từ chương trình VBA thông qua phương thức SendCommand của đối tượng Document tương ứng. Ví dụ sau sẽ tạo ra một hình tròn trong bản vẽ hiện hành của AutoCAD với việc sử dụng lệnh Circle và Zoom của AutoCAD:

```
Sub Tao_Hinh_Tròn()
    ThisDrawing.SendCommand "_Circle" & vbCr & "2,2,0" & vbCr & "4" &
vbCr
    ThisDrawing.SendCommand "_zoom" & vbCr & "a" & vbCr
End Sub
```

Thực chất của phương thức SendCommand là yêu cầu AutoCAD thực thi một lệnh từ dòng lệnh trong AutoCAD. Ký tự vbCr tương đương với việc bấm phím Enter khi thao tác trực tiếp trong AutoCAD.

### 5.1.4. Thu phóng màn hình bản vẽ (zoom)

Thu phóng màn hình bản vẽ trong AutoCAD được thực hiện thông qua các phương thức có trong đối tượng Application. Các phương thức này tương ứng với lệnh zoom trong AutoCAD. Nếu có nhiều bản vẽ đang được mở trong AutoCAD thì các phương thức này chỉ có tác dụng đối với bản vẽ hiện hành.

Dưới đây là các phương thức dùng để thu phóng màn hình bản vẽ trong AutoCAD.

#### ZoomExtents

Phương thức này sẽ phóng màn hình bản vẽ theo vùng bao của tất cả các đối tượng trong bản vẽ, nghĩa là giúp ta có thể quan sát được tất cả các đối tượng hình học hiện đang có với kích thước lớn nhất. Đoạn mã ví dụ sau sẽ thực hiện phương thức ZoomExtents:

```
Application.ZoomExtents
```

#### ZoomAll

Trong chế độ 2D, phương thức này sẽ phóng màn hình bản vẽ theo giới hạn của bản vẽ hoặc theo vùng bao tất cả các đối tượng tùy thuộc vào vùng nào rộng hơn. Còn trong chế độ 3D, phương thức này tương đương với phương thức ZoomExtents.

Đoạn mã sau phóng màn hình bản vẽ sử dụng phương thức ZoomAll:

```
Application.ZoomAll
```

#### ZoomPrevious

Phương thức này sẽ chuyển màn hình bản vẽ về trạng thái trước đó. Khi người dùng thực hiện lệnh Pan hoặc các lệnh liên quan đến thu phóng màn hình bản vẽ, AutoCAD sẽ tự động lưu trạng thái màn hình bản vẽ. Phương thức này có thể khôi phục lại trạng thái màn hình đã được lưu đến 10 cấp.

Đoạn mã sau khôi phục lại trạng thái màn hình bản vẽ sử dụng phương thức ZoomPrevious:

```
Application.ZoomPrevious
```

#### ZoomPickWindow

## GIÁO TRÌNH TỰ ĐỘNG HOÁ THIẾT KẾ CẦU ĐƯỜNG

Phương thức này sẽ phóng màn hình bản vẽ theo một hình chữ nhật do người dùng chọn trên màn hình.

Đoạn mã sau minh họa cách sử dụng phương thức này:

```
Application.ZoomPickWindow
```

### ZoomWindow

Phương thức này thực hiện thu phóng màn hình bản vẽ theo một hình chữ nhật được xác định trước. Cú pháp của phương thức này như sau:

```
Application.ZoomWindow Dưới_Trái, Trên_Phải
```

Tham số	Giải thích
Dưới_Trái	Mảng 3 phần tử kiểu Double, xác định tọa độ điểm ở góc dưới bên trái của hình chữ nhật sẽ thực hiện phóng đại.
Trên_Phải	Mảng 3 phần tử kiểu Double, xác định tọa độ điểm ở góc trên bên phải của hình chữ nhật sẽ thực hiện phóng đại.

Ví dụ sau thực hiện thu phóng màn hình bản vẽ theo hình chữ nhật có tọa độ của các điểm ở góc lần lượt là (1.3, 7.8, 0) và (13.7, -2.6, 0):

```
Sub VD_ZoomWindow()  
  \Khai báo biến để chứa tọa độ các điểm góc  
  Dim point1(0 To 2) As Double  
  Dim point2(0 To 2) As Double  
  \ Gán tọa độ cho các điểm góc  
  point1(0) = 1.3: point1(1) = 7.8: point1(2) = 0  
  point2(0) = 13.7: point2(1) = -2.6: point2(2) = 0  
  \ Thực hiện phương thức ZoomWindow  
  ZoomWindow point1, point2  
End Sub
```

### ZoomScaled

Phương thức này thu phóng màn hình bản vẽ theo một tỉ lệ được xác định trước. Cú pháp của phương thức này như sau:

```
Application.ZoomScaled Scale[, ScaleType]
```

Tham số	Giải thích
Scale	Tham số xác định tỉ lệ thu phóng màn hình bản vẽ.
ScaleType	Tham số tùy chọn, xác định cách thức áp dụng hệ số tỉ lệ. Có thể bằng một trong các hằng số sau:  acZoomScaledAbsolute : tương đối so với vùng vẽ (drawing limits).  acZoomScaledRelative : tương đối so với màn hình bản vẽ hiện hành.  acZoomScaledRelativePSPACE : tương đối so với đơn vị của không gian mô hình.

Ví dụ sau minh họa cách thức sử dụng phương thức ZoomScaled bằng cách phóng màn hình bản vẽ lên 2 lần so với màn hình bản vẽ hiện tại:

```
Sub VD_ZoomScaled()
    Dim tỉ_le As Double
    Dim kiểu_phong_dai As Integer
    tỉ_le = 2
    kiểu_phong_dai = acZoomScaledRelative
    ` Thực hiện phương thức ZoomScaled
    ZoomScaled tỉ_le, kiểu_phong_dai
End Sub
```

### 5.1.5. Nhập dữ liệu người dùng từ dòng lệnh của AutoCAD

Trong một chương trình, giao diện để người sử dụng thao tác với chương trình là một bộ phận rất quan trọng và không thể thiếu. Thông qua giao diện, người sử dụng có thể nhập dữ liệu và điều khiển chương trình hoạt động, còn chương trình, cũng thông qua giao diện, sẽ hướng dẫn cho người dùng cách thao tác và trình bày kết quả thực hiện của nó cho người dùng.

Có nhiều cách để thiết kế giao diện nhập dữ liệu cho chương trình, như sử dụng các hộp thoại chuẩn (như InputBox hoặc MsgBox) hay thông qua hệ thống các hộp thoại người dùng (UserForm). Khi lập trình VBA trong AutoCAD, bởi chương trình sẽ hoạt động dựa trên nền là AutoCAD cho nên việc thiết kế một giao diện cho phép người dùng tương tác với chương trình ngay trong giao diện của AutoCAD là một nhu cầu cần thiết. Hơn nữa điều này được AutoCAD và VBA hỗ trợ thông qua đối tượng Utility (là một thuộc tính của đối tượng Document). Với những phương thức của đối tượng Utility người lập trình có cho phép người sử dụng thao tác với chương trình VBA thông qua dòng lệnh của AutoCAD cũng như màn hình đồ họa của AutoCAD. Các phương thức này sẽ hiển thị một dòng nhắc trên dòng lệnh của AutoCAD và yêu cầu người sử dụng nhập vào nhiều kiểu dữ liệu khác nhau (tùy thuộc vào từng loại phương thức) từ bàn phím hoặc chọn trên màn hình đồ họa của AutoCAD.

Các phương thức để người dùng nhập dữ liệu vào từ bàn phím hay bằng chuột thường có dạng GetXXX, tùy thuộc vào loại dữ liệu mà người lập trình cần lấy. Dưới đây là một số phương thức thường được sử dụng:

#### Prompt

Phương thức này chỉ đơn giản là gửi một đoạn văn bản đến dòng lệnh của AutoCAD và thường được sử dụng để thông báo cho người dùng biết một nội dung nào đó trước hoặc sau một thao tác với chương trình. Cú pháp như sau:

**Utility.Prompt Message**

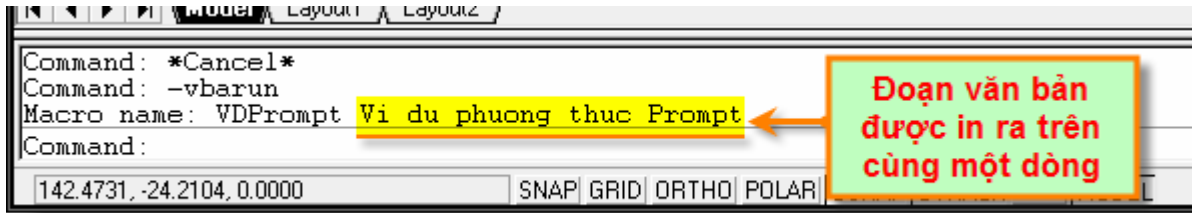
Trong đó Message là đoạn văn bản sẽ được hiển thị trên dòng lệnh của AutoCAD.

Khi gửi một đoạn văn bản đến dòng lệnh của AutoCAD, cần thêm vào ký tự xuống dòng, tránh dòng văn bản cần hiển thị nối vào dòng văn bản đang có trong dòng lệnh. Ví dụ sau sẽ minh họa rõ hơn điều này.

1. Trong VBAIDE, tạo Macro sau:

```
Sub VDPrompt()
    Utility.Prompt ("Vi du phuong thuc Prompt")
End Sub
```

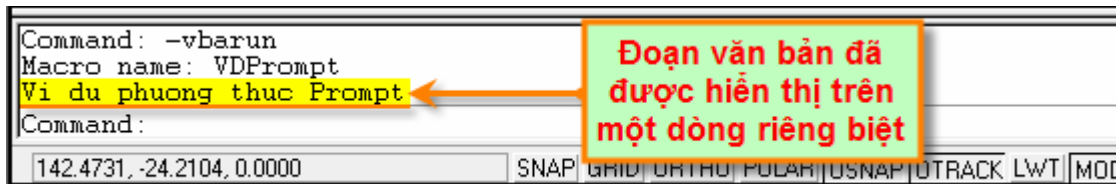
2. Trở về AutoCAD để thực thi Macro bằng cách gọi lệnh `-vbarun`. Lưu ý là sử dụng phím SPACE để kết thúc dòng lệnh, thay vì sử dụng phím ENTER như thông thường. Kết quả hiển thị trên dòng lệnh như sau:



3. Để đoạn văn bản được in ra trên một dòng riêng biệt, thêm vào trước đoạn văn bản hàng số `vbCrLf`, và đoạn mã lệnh trên được chuyển thành:

```
Sub VDPrompt ()
    Utility.Prompt (vbCrLf & "Vi du phuong thuc Prompt")
End Sub
```

4. Thực thi lại Macro trên, đoạn văn bản đã được hiển thị trên một dòng riêng biệt



## GetString

Phương thức này được sử dụng để người dùng nhập vào một chuỗi ký tự. AutoCAD sẽ dừng lại cho đến khi người dùng nhập vào một giá trị nào đó. Cú pháp của phương thức GetString như sau:

```
RetVal = Utility.GetString(HasSpaces[, Prompt])
```

Tham số	Giải thích
HasSpaces	Tham số cho phép người dùng nhập vào dấu cách. Nếu bằng TRUE, người dùng có thể nhập dấu cách trong dòng lệnh, để kết thúc nhập phải nhấn phím ENTER. Nếu bằng FALSE, người dùng không thể nhập dấu cách cho chuỗi ký tự, khi người dùng nhấn phím SPACE hoặc ENTER thì sẽ kết thúc quá trình nhập.
Prompt	Tham số tùy chọn, là chuỗi ký tự sẽ hiện trên dòng lệnh AutoCAD để nhắc người dùng nhập dữ liệu.
RetVal	Là biến kiểu String, chứa giá trị là chuỗi ký tự được người dùng nhập vào. Một điểm cần lưu ý là phương thức này chỉ trả về tối đa 132 ký tự. Nếu người dùng nhập nhiều hơn 132 ký tự, kết quả trả về cho biến RetVal chỉ là 132 ký tự đầu tiên.

Ví dụ sau sẽ minh họa cách sử dụng phương thức GetString:

```
Sub VD_GetString()
    ' Ví dụ minh họa các cách sử dụng phương thức GetString

    Dim returnString As String

    ' Nhắc người dùng nhập
```



```
' Giá trị nhập vào không thể chứa dấu cách
returnString = ThisDrawing.Utility.GetString _
    (False, "Nhập chuỗi (nhấn SPACE hoặc ENTER để kết thúc): ")
MsgBox "Chuỗi vừa nhập là: " & returnString & ""

' Nhắc người dùng nhập
' Giá trị nhập vào có thể chứa dấu cách
returnString = ThisDrawing.Utility.GetString _
    (True, " Nhập chuỗi (nhấn ENTER để kết thúc): ")
MsgBox "Chuỗi vừa nhập là: " & returnString & ""
End Sub
```

### GetInteger, GetReal

Phương thức này được sử dụng khi muốn người dùng nhập một số nguyên (phương thức GetInteger) hoặc một số thực (phương thức GetReal). Cú pháp của các phương thức này như sau:

```
RetVal = Utility.GetInteger ([Prompt])
RetVal = Utility.GetReal ([Prompt])
```

Tham số	Giải thích
Prompt	Tham số tùy chọn, là chuỗi ký tự sẽ hiện trên dòng lệnh AutoCAD để nhắc người dùng nhập dữ liệu.
RetVal	Là biến kiểu Double hoặc Integer (tùy thuộc vào phương thức được sử dụng), chứa giá trị là số người dùng vừa nhập vào.

Nếu người dùng nhập vào một từ khóa hoặc không nhập số mà nhấn ngay phím ENTER để kết thúc nhập liệu, AutoCAD sẽ phát sinh lỗi “*User input keyword.*”.

Ví dụ sau minh họa cách sử dụng các phương thức này:

```
Sub Example_GetReal ()
' Ví dụ sau sử dụng phương thức GetReal và phương thức GetInteger
' để người dùng nhập vào số thực và số nguyên.

Dim returnReal As Double
Dim returnInteger As Integer

' Nhắc người dùng nhập vào số thực,
' sau đó hiển thị kết quả được nhập vào.
returnReal = ThisDrawing.Utility.GetReal("Enter an Real: ")
MsgBox "Số thực vừa được nhập: " & returnReal & vbCrLf & _
    "(Tiếp tục nhập giá trị.)"

' Nhắc người dùng nhập vào số nguyên,
' sau đó hiển thị kết quả được nhập vào.
returnInteger = ThisDrawing.Utility.GetInteger("Nhập số nguyên: ")
MsgBox "Số nguyên vừa được nhập: " & returnInteger
End Sub
```

### GetAngle

## GIÁO TRÌNH TỰ ĐỘNG HOÁ THIẾT KẾ CẦU ĐƯỜNG

Phương thức này được sử dụng khi muốn người lập trình nhập vào một giá trị góc bằng cách nhập giá trị ngay trên dòng lệnh hoặc chọn một góc trên màn hình. Cú pháp của phương thức này như sau:

```
RetVal = Utility.GetAngle([Point][, Prompt])
```

Tham số	Giải thích
Point	Tham số tùy chọn, là mảng số thực có 3 phần tử thể hiện tọa độ đầu tiên của tia tạo nên góc mà người dùng sẽ chọn trên màn hình.
Prompt	Tham số tùy chọn, là chuỗi ký tự sẽ hiện trên dòng lệnh AutoCAD để nhắc người dùng nhập dữ liệu.
RetVal	Là biến kiểu Double chứa giá trị trả về của phương thức GetAngle, là góc mà người dùng đã nhập vào (tính theo Radian).

Khi sử dụng phương thức này, người dùng có thể nhập vào góc (tính bằng độ) tại dòng lệnh. Ngoài ra, người sử dụng có thể sử dụng chuột để vẽ hai điểm xác định tia tạo nên góc cần nhập. Giá trị trả về là góc hợp giữa tia đó và góc cơ sở (được xác định bởi biến hệ thống ANGBASE, giá trị mặc định là 0).



Trong trường hợp không nhập giá trị cho tham số Point, người dùng sẽ phải chọn hai điểm trên màn hình để xác định tia mong muốn. Nếu nhập giá trị cho tham số Point, thì tọa độ có trong tham số Point sẽ được gán cho điểm thứ 1, người dùng chỉ cần xác định điểm thứ 2 trên màn hình mà thôi.

Bảng dưới đây thể hiện giá trị trả về của phương thức GetAngle với các góc nhập vào khác nhau:

Góc người dùng nhập (độ)	Giá trị trả về của phương thức GetAngle
0	0.0
-90	1.5708
180	3.14159
90	4.71239

Nếu người dùng không nhập giá trị nào cả mà nhấn ENTER, AutoCAD sẽ phát sinh lỗi “*User input keyword.*”

Ví dụ sau minh họa cách sử dụng phương thức GetAngle:

```
Sub Example_GetAngle()

Dim retAngle As Double

' Lấy về góc tính bằng radian
retAngle = ThisDrawing.Utility.GetAngle(, "Nhập vào góc: ")
MsgBox "Góc vừa được nhập là: " & retAngle

' Lấy về góc tính bằng radian với tọa độ điểm đầu cho trước
Dim basePnt(0 To 2) As Double
basePnt(0) = 2#: basePnt(1) = 2#: basePnt(2) = 0#
retAngle = ThisDrawing.Utility.GetAngle(basePnt, "Nhập vào góc: ")
MsgBox "Góc vừa được nhập là: " & retAngle

End Sub
```

### GetPoint

Phương thức GetPoint được sử dụng để lấy một điểm do người dùng nhập vào bằng cách nhập tọa độ trực tiếp từ dòng lệnh hoặc chọn điểm trên màn hình. Giá trị trả về của phương thức có kiểu Variant, là một mảng gồm 3 phần tử số thực chứa tọa độ của điểm đã được chọn trong hệ tọa độ WCS. Cú pháp của phương thức như sau:

```
RetVal = Utility.GetPoint([Point][, Prompt])
```

Tham số	Giải thích
Point	Tham số tùy chọn, kiểu Variant, là mảng số thực có 3 phần tử thể hiện tọa độ của điểm tham chiếu của điểm sẽ nhập vào.
Prompt	Tham số tùy chọn, là chuỗi ký tự sẽ hiện trên dòng lệnh AutoCAD để nhắc người dùng nhập dữ liệu.

Nếu tham số tùy chọn Point được gán giá trị, AutoCAD sẽ tạo một đường thẳng tham chiếu nối từ điểm Point đến vị trí hiện tại của con trỏ trên màn hình đồ họa. Đường thẳng này luôn thay đổi theo sự di chuyển của con trỏ, hỗ trợ việc quan sát của người dùng trong quá trình nhập điểm. Sau khi người dùng nhập điểm bằng cách bấm chuột trên màn hình đồ họa tại vị trí mong muốn thì đường thẳng tham chiếu cũng sẽ mất đi.

Nếu người dùng không nhập vào điểm nào mà nhấn ENTER, AutoCAD sẽ phát sinh lỗi “*User input keyword.*”

Ví dụ sau minh họa cách sử dụng phương thức GetPoint và các tham số:

```
Sub Example_GetPoint()
' Ví dụ minh họa cách sử dụng phương thức GetPoint.

Dim returnPnt As Variant

' Nhập điểm và trả về tọa độ của điểm khi không có điểm tham chiếu
returnPnt = ThisDrawing.Utility.GetPoint(, "Nhập một điểm: ")
MsgBox "Tọa độ WCS của điểm: " & returnPnt(0) & ", " & _
returnPnt(1) & ", " & returnPnt(2)
```

```
' Nhập điểm sử dụng điểm tham chiếu
Dim basePnt(0 To 2) As Double
basePnt(0) = 2#: basePnt(1) = 2#: basePnt(2) = 0#
returnPnt = ThisDrawing.Utility.GetPoint(basePnt, "Nhập một điểm: ")
MsgBox "Toa do WCS của điểm: " & returnPnt(0) & ", " & _
returnPnt(1) & ", " & returnPnt(2)
```

```
' Vẽ đường thẳng nối từ điểm tham chiếu đến điểm
' cuối cùng nhập vào
Dim lineObj As AcadLine
Set lineObj = ThisDrawing.ModelSpace.AddLine(basePnt, returnPnt)
ZoomAll

End Sub
```

## GetDistance

Phương thức GetDistance được sử dụng để người dùng nhập vào giá trị khoảng cách. Người dùng có thể nhập một số thực trực tiếp từ dòng lệnh hoặc có thể chọn hai điểm trên màn hình bản vẽ, AutoCAD sẽ tự động trả về giá trị số thực là khoảng cách giữa hai điểm. Cú pháp của phương thức GetDistance như sau:

```
RetVal = Utility.GetDistance([Point][, Prompt])
```

Tham số	Giải thích
Point	Tham số tùy chọn, kiểu Variant, là mảng số thực có 3 phần tử thể hiện tọa độ của điểm cơ sở để tính khoảng cách. Nếu tham số này không có thì người dùng phải chọn hai điểm để xác định khoảng cách.
Prompt	Tham số tùy chọn, là chuỗi ký tự sẽ hiện trên dòng lệnh AutoCAD để nhắc người dùng nhập dữ liệu.
RetVal	Là biến kiểu số thực chứa giá trị trả về của phương thức GetDistance.

Phương thức GetDistance cho phép người dùng nhập một số âm tại dòng nhắc và sẽ trả về một số âm tương ứng. Nhưng khi người dùng chọn điểm trên màn hình bản vẽ, phương thức luôn trả về giá trị tuyệt đối của khoảng cách giữa hai điểm.

Nếu khoảng cách được nhập vào bằng cách chọn điểm trên màn hình, AutoCAD sẽ tạo ra một đường thẳng để giúp người dùng quan sát và đường thẳng này sẽ mất đi sau khi người dùng nhập xong khoảng cách. Nếu không nhập giá trị cho tham số Point, người dùng sẽ phải xác định hai điểm trên màn hình để xác định khoảng cách. Nếu gán giá trị cho tham số Point, người dùng chỉ cần chọn thêm một điểm trên màn hình, giá trị khoảng cách sẽ được tính từ điểm truyền cho tham số Point và điểm cho người dùng chọn.

Theo mặc định của AutoCAD, các điểm nhập vào có tọa độ không gian gồm đầy đủ 3 thành phần (x, y, z) nên khoảng cách giữa hai điểm là khoảng cách trong không gian. Người dùng có thể yêu cầu AutoCAD chỉ tính khoảng cách phẳng bằng cách thực hiện phương thức InitializeUserInput trước khi thực hiện phương thức GetDistance với tham số OptionBits tương ứng để AutoCAD bỏ qua thành phần tọa độ z.

Nếu người dùng không nhập giá trị hoặc điểm nào cả mà nhấn ENTER thì AutoCAD sẽ phát sinh lỗi “User input keyword.”

Ví dụ dưới đây minh họa cách sử dụng phương thức GetDistance:

```

Sub Example_GetDistance()
' Ví dụ minh họa cách sử dụng phương thức GetDistance.

Dim returnDist As Double

' Nhập và trả về giá trị khoảng cách, có sử dụng dòng nhắc
returnDist = ThisDrawing.Utility.GetDistance(, "Nhập khoảng cách: ")

MsgBox "Khoang cach vua nhap la: " & returnDist & vbCrLf & _
"(Nhập giá trị tiếp theo cơ số dùng điểm cơ số.)"

' Nhập và trả về giá trị khoảng cách
' Có sử dụng dòng nhắc và điểm cơ số
Dim basePnt(0 To 2) As Double
basePnt(0) = 2#: basePnt(1) = 2#: basePnt(2) = 0#
returnDist = ThisDrawing.Utility.GetDistance(basePnt, _
"Nhập khoảng cách: ")

MsgBox "Khoang cach vua nhap la: " & returnDist

End Sub

```

### GetEntity

Phương thức GetEntity được sử dụng để lấy **một** đối tượng của AutoCAD bằng cách cho phép người dùng chọn trực tiếp bằng chuột trên màn hình đồ họa. Cú pháp của phương thức như sau:

```
Utility.GetEntity Object, PickedPoint[, Prompt]
```

Tham số	Giải thích
Object	Tham số trả về đối tượng được người dùng chọn
Pickedpoint	Tham số kiểu Variant, trả về mảng số thực có 3 phần tử thể hiện tọa độ của điểm mà người dùng kích chuột để chọn đối tượng.
Prompt	Tham số tùy chọn, là chuỗi ký tự sẽ hiện trên dòng lệnh AutoCAD để nhắc người dùng nhập dữ liệu.

Phương thức GetEntity yêu cầu người dùng chọn đối tượng bằng cách kích chuột trên màn hình bản vẽ. Nếu người dùng chọn một đối tượng, đối tượng đó sẽ được trả về thông qua tham số Object và tọa độ của điểm mà người dùng chọn sẽ được trả về trong tham số PickedPoint. Nếu điểm mà người dùng kích chuột không phải đối tượng thì phương thức này sẽ làm phát sinh lỗi.

Với phương thức này, người dùng còn có thể chọn nhanh đối tượng được vẽ sau cùng nhất bằng cách nhập ký tự “L” hoặc “I” tại dòng lệnh AutoCAD. Khi dùng lệnh “L” này, nếu đối tượng được vẽ cuối cùng không nhìn thấy trên màn hình bản vẽ (đối tượng có thuộc tính Visible=FALSE) hoặc đối tượng đó đang nằm trong một lớp đã bị đóng băng (FrozenLayer) thì đối tượng cuối cùng nhất được vẽ nằm trên một lớp bình thường sẽ được chọn. Tuy nhiên, cách ứng xử này có thể khác nhau trong từng phiên bản của AutoCAD, chẳng hạn như đối với AutoCAD 2007, với lệnh “L”, phương thức GetEntity có thể trả về một đối tượng không được nhìn thấy trên màn hình bản vẽ.

**CHÚ Ý** Với phương thức GetEntity, tại một thời điểm, người dùng chỉ có thể chọn được một đối tượng.

Ví dụ dưới đây minh họa cách sử dụng phương thức GetEntity. Đối tượng được lựa chọn sẽ được tạm thời chuyển sang màu đỏ để người dùng dễ nhận thấy trước khi chuyển sang lựa chọn đối tượng khác (trước khi thực hiện ví dụ cần tạo sẵn một số đối tượng hình học trong bản vẽ hiện hành của AutoCAD). Ví dụ này còn thực hiện bẫy lỗi phát sinh khi lựa chọn đối tượng.

```
Sub VD_GetEntity()  
  
    Dim returnObj As AcadObject  
  
    Dim basePnt As Variant  
  
    On Error Resume Next  
  
    ' Trong ví dụ này, AutoCAD chờ người dùng lựa chọn đối tượng  
    ThisDrawing.Utility.GetEntity returnObj, basePnt, "Chọn đối tượng:"  
  
    If Err <> 0 Then  
        Err.Clear  
        MsgBox "Bạn không chọn đối tượng. Tạm biệt."  
        Exit Sub  
    Else  
        returnObj.Color = acRed  
        returnObj.Update  
        MsgBox "Kiểu đối tượng là: " & returnObj.EntityName  
        MsgBox "tại vị trí " & basePnt(0) & ", " & basePnt(1)  
        returnObj.Color = acByLayer  
        returnObj.Update  
    End If  
End Sub
```

### 5.1.6. Thiết lập biến hệ thống

AutoCAD sử dụng các biến hệ thống dùng để điều khiển các hoạt động của chính nó, chẳng hạn như chế độ bắt điểm, chế độ lưới, điều kiện cách thực hiện của các lệnh,... Phần này sẽ giới thiệu cách thức đọc và thiết lập các biến hệ thống cho AutoCAD thông qua các phương thức GetVariable và SetVariable cũng như một số biến hệ thống thường dùng trong AutoCAD.

Đối với mỗi phiên bản của AutoCAD, các biến hệ thống có thể khác nhau, vì vậy cần phải nghiên cứu trong tài liệu đi kèm với phiên bản AutoCAD đang sử dụng để có được các thông tin cụ thể về các biến hệ thống.

Một điểm cần lưu ý khi thao tác với các biến hệ thống của AutoCAD là phạm vi tác dụng của biến hệ thống. Có thể chia thành hai loại sau:

- ◆ Loại biến có tác dụng với toàn bộ ứng dụng AutoCAD: với loại biến này, khi người thay đổi giá trị của biến, tất cả các bản vẽ đang được mở sẽ chịu tác động do sự thay đổi của biến này. Các biến kiểu này được lưu trong bản thân chương trình AutoCAD.
- ◆ Loại biến có tác dụng với một bản vẽ AutoCAD: với loại biến này, khi người dùng thay đổi giá trị của biến, chỉ có bản vẽ hiện hành (bản vẽ nơi thực hiện thao tác thay đổi giá trị của biến) là chịu tác động do sự thay đổi của biến. Các biến hệ thống kiểu này được lưu ngay bên trong tệp bản vẽ.

### Phương thức GetVariable

Phương thức này dùng để lấy về giá trị hiện hành của một biến hệ thống trong AutoCAD. Cú pháp của phương thức này như sau:

```
RetVal = object.GetVariable (Name)
```

Tham số	Giải thích
Name	Tham số kiểu String xác định tên biến hệ thống cần lấy giá trị (không phân biệt chữ hoa/chữ thường). Nếu tên biến hệ thống nhập vào không đúng sẽ phát sinh lỗi khi thực thi chương trình.
object	Đối tượng kiểu Document, nơi thực hiện phương thức GetVariable.
RetVal	Là biến kiểu Variant chứa giá trị trả về của biến hệ thống.

Ví dụ sau minh họa các sử dụng phương thức này bằng cách thực hiện lấy giá trị của biến hệ thống MIRRTEXT:

```
Sub VD_GetVariable()
    ' Ví dụ sau hiển thị giá trị hiện tại của
    ' biến hệ thống MIRRTEXT.

    Dim strTenBien As String
    Dim KetQua As Variant

    strTenBien = "MIRRTEXT"
    KetQua = ThisDrawing.GetVariable(strTenBien)
    MsgBox (strTenBien & " = " & KetQua)
End Sub
```

### Phương thức SetVariable

Phương thức này dùng để thiết lập giá trị cho biến hệ thống trong AutoCAD. Cú pháp của phương thức này như sau:

**object.SetVariable Name, Value**

Tham số	Giải thích
Name	Tham số kiểu String xác định tên biến hệ thống cần lấy giá trị (không phân biệt chữ hoa/chữ thường). Nếu tên biến hệ thống nhập vào không đúng sẽ phát sinh lỗi khi thực thi chương trình.
Value	Tham số kiểu Variant, xác định giá trị cần gán cho biến hệ thống có tên trong tham số Name ở trên.
object	Đối tượng kiểu Document, nơi thực hiện phương thức SetVariable.

Biến hệ thống trong AutoCAD rất đa dạng với nhiều kiểu dữ liệu khác nhau: số thực, số nguyên, chuỗi,... Chính vì vậy, khi gán giá trị cho biến hệ thống, cần phải chú ý sao cho kiểu dữ liệu của biến hệ thống cần gán và kiểu giá trị của tham số Value phải tương thích nhau, nếu không sẽ làm phát sinh lỗi khi thực thi chương trình.

Ví dụ sau minh họa cách thức gán giá trị cho biến hệ thống sử dụng phương thức SetVariable:

```
Sub Example_SetVariable()
    ' Ví dụ sau gán giá trị cho nhiều biến hệ thống khác nhau,
    ' mỗi biến có một kiểu dữ liệu khác nhau.
    Dim TenBien As String
    Dim GiaTri As Variant

    ' Gán giá trị biến MIRRTEXT (kiểu số nguyên) bằng 1.
    ' Chú ý rằng cần phải gán giá trị thích hợp với
    ' kiểu dữ liệu của biến hệ thống.
    Dim intData As Integer
```

```

TenBien = "MIRRTEXT"
intData = 0
GiaTri = intData           ' Kiểu Integer
ThisDrawing.SetVariable TenBien, GiaTri

' Kiểm tra giá trị đã gán sử dụng phương thức GetVariable
GiaTri = ThisDrawing.GetVariable(TenBien)

MsgBox (TenBien & " = " & GiaTri)

' Gán giá trị biến LTSCALE (kiểu số thực) bằng 1.5
Dim dataDouble As Double
TenBien = "LTSCALE"
dataDouble = 1.5
GiaTri = dataDouble       ' Kiểu Double
ThisDrawing.SetVariable TenBien, GiaTri
' Kiểm tra giá trị đã gán sử dụng phương thức GetVariable
GiaTri = ThisDrawing.GetVariable(TenBien)
MsgBox (TenBien & " = " & GiaTri)

' Gán giá trị biến INSBASE (kiểu mảng chứa tọa độ) bằng (1.0,1.0,0)
Dim arrayData3D(0 To 2) As Double
TenBien = "INSBASE"
arrayData3D(0) = 1#: arrayData3D(1) = 1#: arrayData3D(2) = 0
GiaTri = arrayData3D      ' Kiểu mảng chứa tọa độ
điểm
ThisDrawing.SetVariable TenBien, GiaTri
' Kiểm tra giá trị đã gán sử dụng phương thức GetVariable
GiaTri = ThisDrawing.GetVariable(TenBien)
MsgBox (TenBien & " = " & GiaTri(0) & ", " & GiaTri(1) & ", " & GiaTri(2))

End Sub

```

## Các biến hệ thống thường dùng

Để hiển thị được đầy đủ tất cả các biến hệ thống cũng như giá trị hiện thời của chúng, có thể thực hiện theo các bước sau:

1. Tại dòng lệnh của AutoCAD, nhập lệnh **setvar**.
2. Tại dòng nhắc “*Enter Variable Name*”, nhập dấu ?
3. Tại dòng nhắc “*Enter Variable(s) to List*”, nhấn **ENTER**

Bảng dưới đây giới thiệu các biến hệ thống thường dùng trong AutoCAD:

Biến hệ thống	Giải thích
ANGBASE	Thiết lập góc cơ sở, là góc hợp với phương X. Mặc định giá trị này bằng 0.
AUPREC	Thiết lập số chữ số sau dấu phẩy của đơn vị đo góc.
DIMTIH	Xác định vị trí chữ ghi kích thước. Nếu bằng 0: song song với đường ghi kích thước; bằng 1: nằm ngang. Mặc định giá trị này bằng 1.
FILLETRAD	Xác định bán kính vượt cong mặc định khi dùng với lệnh Fillet
INSBASE	Tọa độ điểm chèn mặc định.
CELTSCALE	Thiết lập tỷ lệ kiểu đường cho các đối tượng mới tạo
LTSCALE	Thiết lập tỷ lệ kiểu đường cho tất cả các đối tượng



MIRRTEXT	Xác định cách thức khi lấy đối xứng đối với văn bản. Bằng 0: giữ nguyên chiều văn bản; bằng 1: đổi chiều văn bản.
TILEMODE	Xác định không gian hiện hành là không gian mô hình hay không gian in. Bằng 0: không gian in; bằng 1: không gian mô hình.
ZOOMFACTOR	Xác định tỷ lệ phần trăm thu/phóng bản vẽ khi nhấn phím chuột giữa.
TEXTFILL	Điều khiển cách hiển thị văn bản TrueType. Bằng 0: chỉ vẽ đường biên văn bản; bằng 1: vẽ cả đường biên và tô đầy văn bản.

## 5.2. Tạo mới đối tượng hình học

Mô hình đối tượng trong AutoCAD thực chất là sự mô tả lại hầu như tất cả các đối tượng mà người dùng có thể tạo ra trong AutoCAD theo cách vẽ thông thường, cho nên để lập trình tạo ra các đối tượng hình học bằng VBA thì người dùng cần phải thông thuộc cách tạo ra đối tượng đó bằng lệnh thông thường trực tiếp trong AutoCAD.

Trong AutoCAD, để hỗ trợ người dùng thao tác nhanh, một đối tượng hình học có thể được tạo ra theo nhiều phương thức khác nhau, chẳng hạn như khi tạo đường tròn trong AutoCAD, người dùng có thể tạo theo 4 cách khác nhau:

- ◆ Xác định tâm và bán kính,
- ◆ Xác định 2 điểm tạo nên đường kính đường tròn,
- ◆ Xác định ba điểm ngoại tiếp đường tròn,
- ◆ Xác định hai đường tang và bán kính.

Hoặc để tạo một cung tròn, trong AutoCAD, người dùng có tới 11 phương thức để lựa chọn như hình bên.

Tuy nhiên, với VBA trong AutoCAD, mỗi đối tượng chỉ có thể được tạo bằng một phương thức với một loại thông số nhất định, ví dụ như đối với đường tròn, người lập trình chỉ có thể tạo ra với các thông số là vị trí tâm và bán kính của đường tròn.

Hầu hết các đối tượng hình học trong AutoCAD, tuy khác nhau về hình dáng, nhưng cách tạo ra chúng bằng VBA lại tương tự nhau, cho nên trong phần này chỉ giới thiệu cách thức tạo ra một số đối tượng hình học chính trong AutoCAD, bao gồm:

- ◆ Đối tượng Point;
- ◆ Đối tượng dạng đường cong: Arc, Circle;
- ◆ Đối tượng văn bản: Text;
- ◆ Các đối tượng dạng đường có chiều dài hữu hạn như Line, Polyline...

3 Points
Start, Center, End
Start, Center, Angle
Start, Center, Length
Start, End, Angle
Start, End, Direction
Start, End, Radius
Center, Start, End
Center, Start, Angle
Center, Start, Length
Continue

### 5.2.1. Xác định nơi chứa đối tượng

AutoCAD nhóm các đối tượng hình học trong tập đối tượng ModelSpace, PaperSpace và trong đối tượng Block. Tuy nhiên, thường được sử dụng nhất là hai tập đối tượng ModelSpace và PaperSpace:

- ◆ ModelSpace (không gian mô hình) là một phần của bản vẽ, là nơi để người dùng tạo các đối tượng hình học để tạo nên mô hình hoặc bản vẽ mà người dùng dự định thiết kế. Hầu hết tất cả các thao tác xây dựng bản vẽ đều được thực hiện trên không gian mô hình. Trong AutoCAD, chỉ có một không gian mô hình, tương ứng với không gian mô hình này chính là thẻ Model nằm ở góc dưới màn hình bản vẽ trong AutoCAD.

- ✦ PaperSpace (không gian in) cũng cho phép chứa các đối tượng hình học như trong không gian mô hình, tuy nhiên mục đích chính của không gian in là để phục vụ cho quá trình sắp xếp bản vẽ và in ấn. Không gian in thường chứa các khung nhìn theo một tỷ lệ định trước thể một phần của bản vẽ trong không gian mô hình, hoặc các bảng biểu, ghi chú,... Trong AutoCAD, người dùng có thể tạo nhiều không gian in khác nhau, mỗi không gian in tương ứng với một thẻ Layout nằm ở góc dưới màn hình bản vẽ trong AutoCAD. Để truy cập đến các không gian in có trong bản vẽ, có thể sử dụng tập đối tượng Layouts có trong đối tượng kiểu Document.

Tại một thời điểm, trong AutoCAD chỉ có một không gian là hiện hành, có thể là không gian mô hình hoặc không gian in. Để xác định xem không gian nào là không gian hiện hành, người lập trình có thể sử dụng thuộc tính ActiveSpace có trong đối tượng kiểu Document. Thuộc tính này chỉ nhận giá trị là 2 hằng số sau:

Hằng số	Giá trị tương ứng
acModelSpace	1
acPaperSpace	0

Ví dụ sau sẽ hiển thị thông báo tương ứng với không gian hiện hành của AutoCAD:

```
Sub VD_ActiveSpace()  
  If ThisDrawing.ActiveSpace = acModelSpace Then  
    MsgBox "Không gian hiện hành là không gian mô hình."  
  Else  
    MsgBox "Không gian hiện hành là không gian in."  
  End If  
End Sub
```

Ngoài ra người dùng còn có thể chuyển đổi giữa không gian in và không gian mô hình bằng cách gán giá trị cho thuộc tính ActiveSpace. Đoạn mã sau sẽ thực hiện thao tác này:

```
Sub VD_ChuyenKhongGian()  
  With ThisDrawing  
    If .ActiveSpace = acModelSpace Then  
      .ActiveSpace = acPaperSpace  
    Else  
      .ActiveSpace = acModelSpace  
    End If  
  End With  
End Sub
```

Hoặc đơn giản hơn, chỉ cần sử dụng một dòng lệnh sau:

```
Sub VD_ChuyenKhongGian()  
  ThisDrawing.ActiveSpace = (ThisDrawing.ActiveSpace + 1) Mod 2  
End Sub
```

---

**GỢI Ý** Có thể chuyển đổi giữa các không gian bằng cách gán giá trị cho biến hệ thống TILEMODE. Nếu TILEMODE=1, không gian mô hình sẽ là không gian hiện hành. Nếu bằng 0, không gian in sẽ là không gian hiện hành.

---

### 5.2.2. Khai báo và tạo đối tượng hình học

Tất cả các đối tượng trong AutoCAD (kể cả đối tượng hình học và phi hình học) đều có thể được khai báo trong VBA theo dạng thức Acad<TênĐốiTượng>. Chẳng hạn như đối tượng

đường thẳng – Line thì đối tượng tương ứng trong VBA sẽ có kiểu là AcadLine. Ví dụ sau minh họa cách khai báo một đối tượng đường tròn trong VBA:

```
Dim CircleObj as AcadCircle
```

Người dùng có thể tạo mới đối tượng hình học trong không gian mô hình hoặc trong không gian in. Để tạo đối tượng mới, sử dụng phương thức AddXXX có trong tập đối tượng ModelSpace và PaperSpace, trong đó xxx là tên của loại đối tượng hình học cần tạo. Cú pháp như sau:

```
Set Biến_đối_tượng = Object.AddXXX(Danh_sách_tham_số)
```

Trong đó, Object là tập đối tượng ModelSpace hoặc PaperSpace.

Mỗi phương thức AddXXX sẽ trả về một đối tượng tham chiếu đến đối tượng vừa mới được tạo, vì vậy bắt buộc phải sử dụng câu lệnh Set trong khi tạo đối tượng và biến\_đối\_tượng phải có kiểu phù hợp với đối tượng trả về của phương thức AddXXX.

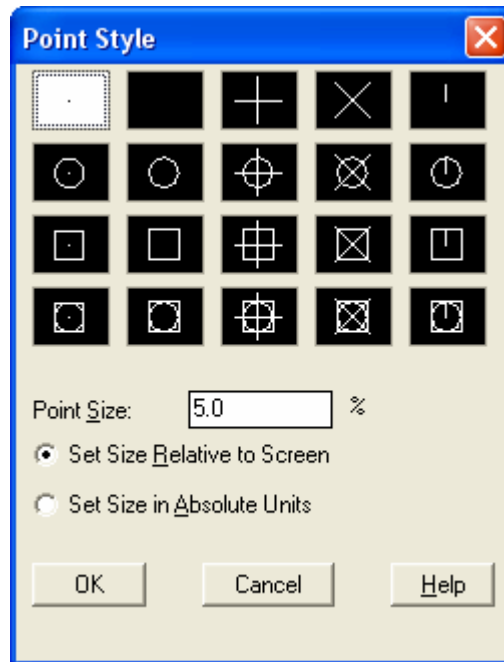
Lấy ví dụ khi muốn tạo mới một đường tròn trong không gian mô hình, có thể sử dụng mẫu sau:

```
Dim CircleObj as AcadCircle  
Set Circle = ThisDrawing.ModelSpace.AddCircle(CenterPoint, Radius)
```

Sau khi tạo mới (hoặc hiệu chỉnh) đối tượng, thì kết quả hiển thị trên bản vẽ sẽ không được cập nhật ngay cho đến khi gọi phương thức Update của bản thân đối tượng đó, hoặc phương thức Update của đối tượng Application hoặc phương thức Regen của đối tượng Document. Trong một số trường hợp, AutoCAD cũng có tự động cập nhật ngay khi kết thúc Macro. Tuy nhiên, để chắc chắn, sau khi tạo mới (hoặc hiệu chỉnh) đối tượng, nên chủ động cập nhật lại những thay đổi này.

### 5.2.3. Tạo đối tượng Point

Đối tượng Point đôi khi cũng rất hữu dụng, chẳng hạn như để tạo một nút hoặc là một điểm tham chiếu để từ đó ta tiến hành bắt điểm hoặc thực hiện lệnh Offset. Ngoài ra, khi thiết lập kiểu và kích thước cho đối tượng Point, người dùng có thể sử dụng nó để trình bày bản vẽ một cách hiệu quả.



### Tạo đối tượng Point

Sử dụng phương thức `AddPoint` để thêm một điểm tại vị trí mong muốn. Cú pháp như sau:

```
SetRetVal = object.AddPoint(Point)
```

Tham số	Giải thích
Point	Là tham số đầu vào kiểu Variant hoặc mảng 3 phần tử kiểu Double, chứa tọa độ của vị trí tạo đối tượng Point.
RetVal	Đối tượng kiểu Point, chứa tham chiếu đến đối tượng vừa mới được tạo.

### Thay đổi kiểu hiển thị của đối tượng Point

Sau khi tạo đối tượng Point, người lập trình có thể thay đổi kiểu hiển thị của điểm bằng cách thiết lập giá trị cho các biến hệ thống `PDMODE` và `PDSIZE`.

Đoạn mã sau sẽ tạo một đối tượng Point trong không gian mô hình ở tọa độ (5, 5, 0) và sau đó thay đổi kiểu hiển thị của đối tượng Point bằng cách cập nhật lại giá trị biến hệ thống `PDMODE` và `PDSIZE`.

```
Sub VD_TaoDTPoint()
    Dim pointObj As AcadPoint
    Dim ToaDo(0 To 2) As Double

    ' Xác định vị trí vẽ điểm
    ToaDo (0) = 5#: ToaDo (1) = 5#: ToaDo (2) = 0#

    ' Tạo điểm mới
    Set pointObj = ThisDrawing.ModelSpace.AddPoint(ToaDo)
    ThisDrawing.SetVariable "PDMODE", 34
    ThisDrawing.SetVariable "PDSIZE", 1
    ZoomAll
End Sub
```

### 5.2.4. Tạo đối tượng dạng đường thẳng

Đường thẳng là đối tượng hình học cơ bản hay được dùng nhất trong AutoCAD. Nhìn chung, ta có thể vẽ các đường thẳng bằng cách nhập vào tham số là tọa độ của các điểm của đường thẳng.

Để tạo một đường thẳng, có thể sử dụng một trong những phương thức sau:

Phương thức	Giải thích
AddLine	Tạo đường thẳng đi qua hai điểm.
AddLightweightPolyline	Tạo đường đa tuyến 2D.
AddMLine	Tạo đường đa tuyến nét đôi.
Add3DPoly	Tạo đường đa tuyến 3D.

### Tạo đối tượng Line

Phương thức AddLine sẽ tạo đối tượng Line, là một đoạn thẳng đi qua hai điểm:

```
SetRetVal = object.AddLine(StartPoint, EndPoint)
```

Tham số	Giải thích
StartPoint, EndPoint	Là tham số đầu vào kiểu variant hoặc mảng 3 phần tử kiểu Double, chứa tọa độ điểm đầu và điểm kết thúc của đoạn thẳng.
RetVal	Đối tượng kiểu Line, tham chiếu đến đoạn thẳng vừa mới được tạo.

Ví dụ sau tạo một đoạn thẳng trong không gian mô hình với tọa độ điểm đầu và điểm cuối là (1, 1, 0) và (5, 5, 0):

```
Sub Example_AddLine()
    Dim lineObj As AcadLine
    Dim diemDau(0 To 2) As Double
    Dim diemCuoi(0 To 2) As Double

    ' Định điểm đầu và điểm cuối của đoạn thẳng
    diemDau(0) = 1#: diemDau(1) = 1#: diemDau(2) = 0#
    diemCuoi(0) = 5#: diemCuoi(1) = 5#: diemCuoi(2) = 0#

    ' Tạo đoạn thẳng trong không gian mô hình
    Set lineObj = ThisDrawing.ModelSpace.AddLine(diemDau, diemCuoi)
    ZoomAll
End Sub
```

Còn trong ví dụ sau, đoạn thẳng sẽ được tạo ra từ hai điểm bất kỳ do người dùng chọn trên màn hình. Đoạn thẳng này sẽ được tạo ra trên không gian mô hình hoặc không gian in, tùy thuộc vào không gian nào là hiện hành.

```
Public Sub TestAddLine()
    Dim diemDau As Variant
    Dim diemCuoi As Variant
    Dim objEnt As AcadLine
    On Error Resume Next
    ' Lấy tọa độ điểm đầu và điểm cuối do người dùng nhập
    diemDau = ThisDrawing.Utility.GetPoint _
        (, vbCr & "Chon diem dau: ")
    diemCuoi = ThisDrawing.Utility.GetPoint _
        (diemDau, vbCr & "Chon diem cuoi: ")
    ' Vẽ đối tượng
    If ThisDrawing.ActiveSpace = acModelSpace Then
```

```

    Set objEnt = ThisDrawing.ModelSpace.AddLine(diemDau, diemCuoi)
Else
    Set objEnt = ThisDrawing.PaperSpace.AddLine(diemDau, diemCuoi)
End If
' Cập nhật đối tượng để hiển thị trên màn hình bản vẽ
objEnt.Update
End Sub

```

## Tạo đối tượng LWPolyline, Polyline

LWPolyline là đối tượng dùng để biểu diễn đường đa tuyến “phẳng” và do đó chỉ dùng để thể hiện các đối tượng trong không gian 2D. Điều này giúp cho dữ liệu của đối tượng LWPolyline gọn nhẹ hơn và các thao tác đồ họa sẽ thực hiện nhanh hơn. Đó cũng chính là lý do tại sao đối tượng này là có tên là “LightWeight - Nhẹ”.

Để tạo đối tượng LWPolyline, sử dụng phương thức AddLightweightPolyline. Cú pháp phương thức này như sau:

```
SetRetVal = object.AddLightweightPolyline(VerticesList)
```

Tham số	Giải thích
VerticesList	Tham số đầu vào kiểu Variant hoặc mảng kiểu Double, chứa tọa độ các đỉnh của đa tuyến.
RetVal	Đối tượng kiểu LWPolyline, tham chiếu đến đa tuyến 2D vừa mới được tạo.

Khi sử dụng phương thức AddLightweightPolyline, tham số VerticesList sẽ chứa tọa độ các đỉnh của đa tuyến 2D. Tọa độ của mỗi đỉnh được biểu diễn bằng hai thành phần x và y, nên các thành phần của mảng được bố trí theo dạng (p1x, p1y, p2x, p2y, ...). Vì vậy số phần tử của tham số VerticesList luôn là bội số của 2 và đương nhiên, tối thiểu cần phải có 4 phần tử (hai điểm) để có thể tạo được đối tượng LWPolyline.

**CHÚ Ý** Mỗi đỉnh của LWPolyline chỉ có hai tọa độ x và y, không có thông số về cao độ. Thay vào đó, người lập trình có thể gán cao độ chung cho toàn bộ đường đa tuyến phẳng này bằng cách thay giá trị cao độ thích hợp cho thuộc tính Elevation.

Đoạn mã sau sẽ tạo một đường đa tuyến phẳng gồm có 5 đỉnh trong không gian mô hình:

```

Sub Example_AddLightWeightPolyline()
    Dim plineObj As AcadLWPolyline
    Dim points(0 To 9) As Double

    ' Xác định các đỉnh của đa tuyến phẳng
    points(0) = 1: points(1) = 1           ' Tọa độ đỉnh 1
    points(2) = 1: points(3) = 2           ' Tọa độ đỉnh 2
    points(4) = 2: points(5) = 3           ' Tọa độ đỉnh 3
    points(6) = 3: points(7) = 2           ' Tọa độ đỉnh 4
    points(8) = 4: points(9) = 4           ' Tọa độ đỉnh 5

    ' Tạo đối tượng LWPolyline trong không gian mô hình
    Set plineObj = ThisDrawing.ModelSpace.AddLightWeightPolyline(points)
    ZoomAll
End Sub

```

Ngoài ra, với VBA trong AutoCAD, người dùng cũng có thể tạo đường đa tuyến phẳng với phương thức `AddPolyline`. Cú pháp của phương thức này tương tự như của phương thức `AddLightweightPolyline`:

```
SetRetVal = object.AddPolyline(VerticesList)
```

Tuy nhiên, đối tượng trả về `RetVal` là đối tượng kiểu `Polyline`. Tham số `VerticesList` cũng chứa tọa độ các đỉnh của đa tuyến, nhưng mỗi đỉnh sẽ có 3 thành phần, hai thành phần đầu là tọa độ ( $x, y$ ) thành phần thứ 3 sẽ không được sử dụng.

Khi muốn tạo đường đa tuyến phẳng, nên sử dụng phương thức `AddLightweightPolyline`, vì đối tượng `LWPolyline` đã tối ưu hoá cho quá trình hiển thị và lưu trữ trong bản vẽ.

### Tạo đối tượng 3DPolyline

Phương thức `Add3DPoly` sẽ tạo một đường đa tuyến 3D dựa trên tọa độ các đỉnh được truyền vào. Cú pháp của phương thức này như sau:

```
SetRetVal = object.Add3DPoly(VerticesList)
```

Tham số	Giải thích
VerticesList	Tham số đầu vào kiểu Variant hoặc mảng kiểu Double, chứa tọa độ các đỉnh của đa tuyến.
RetVal	Đối tượng kiểu 3DPolyline, tham chiếu đến đa tuyến 3D vừa mới được tạo.

Khi sử dụng phương thức `Add3DPoly`, tham số `VerticesList` sẽ chứa tọa độ các đỉnh của đa tuyến. Tọa độ của mỗi đỉnh được biểu diễn bằng ba thành phần  $x, y$  và  $z$  nên các thành phần của mảng được bố trí theo dạng ( $p1x, p1y, p1z, p2x, p2y, p2z...$ ). Vì vậy số phần tử của tham số `VerticesList` luôn là bội số của 3 và đương nhiên, tối thiểu cần phải có 6 phần tử (hai điểm) để có thể tạo được đối tượng `3DPolyline`.

Đoạn mã sau sẽ tạo một đường đa tuyến có 5 đỉnh trong không gian mô hình:

```
Sub VD_AddPolyline()

    Dim plineObj As Acad3DPolyline
    Dim points(0 To 14) As Double

    ' Xác định cách đỉnh của đa tuyến 3D
    points(0) = 1: points(1) = 1: points(2) = 0      ' Tọa độ điểm 1
    points(3) = 1: points(4) = 2: points(5) = 10    ' Tọa độ điểm 2
    points(6) = 2: points(7) = 3: points(8) = 30    ' Tọa độ điểm 3
    points(9) = 3: points(10) = 2: points(11) = 0   ' Tọa độ điểm 4
    points(12) = 4: points(13) = 4: points(14) = 8  ' Tọa độ điểm 5

    ' Tạo đường đa tuyến 3D trong không gian mô hình
    Set plineObj = ThisDrawing.ModelSpace.Add3DPoly (points)
    ZoomAll
End Sub
```

### 5.2.5. Tạo đối tượng dạng đường cong

## GIÁO TRÌNH TỰ ĐỘNG HOÁ THIẾT KẾ CẦU ĐƯỜNG

VBA trong AutoCAD cho phép tạo nhiều loại đối tượng dạng đường cong khác nhau, trong đó có hai loại đối tượng rất phổ biến là Circle – đường tròn và Arc – cung tròn. Tất cả các phương thức này đều tạo đối tượng trong mặt phẳng XY.

### Tạo đối tượng Circle

Để tạo đối tượng Circle, sử dụng phương thức AddCircle. Phương thức này sẽ tạo một đường tròn dựa trên thông số tâm và bán kính của đường tròn:

```
SetRetVal = object.AddCircle(Center, Radius)
```

Tham số	Giải thích
Center	Tham số đầu vào kiểu Variant hoặc mảng 3 phần tử kiểu Double, chứa tọa độ tâm của đường tròn.
Radius	Tham số đầu vào kiểu Double, là bán kính của đường tròn sẽ được tạo.
RetVal	Đối tượng kiểu Circle, tham chiếu đến đường tròn vừa mới được tạo.

Đoạn mã sau tạo một đường tròn bán kính bằng 5 và tọa độ tâm là (1, 2, 0) trong không gian mô hình:

```
Sub Example_AddCircle()  
    Dim circleObj As AcadCircle  
    Dim centerPoint(0 To 2) As Double  
    Dim radius As Double  
  
    ' Xác định tâm và bán kính của đường tròn  
    centerPoint(0) = 1#: centerPoint(1) = 2#: centerPoint(2) = 0#  
    radius = 5#  
  
    ' Tạo đối tượng Circle trong không gian mô hình  
    Set circleObj = ThisDrawing.ModelSpace.AddCircle(centerPoint,  
radius)  
    ZoomAll  
End Sub
```

Hoặc tâm và bán kính của đường tròn có thể được nhập vào:

```
Sub VD_AddCircle()  
    Dim varCenter As Variant  
    Dim dblRadius As Double  
    Dim objEnt As AcadCircle  
  
    On Error Resume Next  
    ' Lấy các thông số do người dùng nhập vào  
    With ThisDrawing.Utility  
        varCenter = .GetPoint(, vbCr & "Chọn tâm đường tròn: ")  
        dblRadius = .GetDistance(varCenter, vbCr & "Nhập bán kính: ")  
    End With  
  
    ' Tạo đối tượng Circle trong không gian mô hình  
    Set objEnt = ThisDrawing.ModelSpace.AddCircle(varCenter, dblRadius)  
    objEnt.Update  
End Sub
```

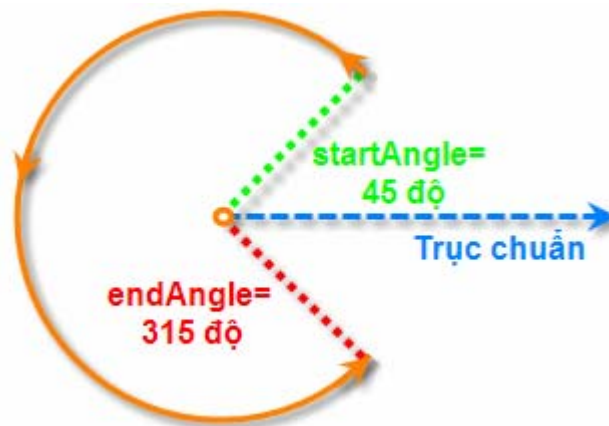


### Tạo đối tượng Arc

Để tạo đối tượng Arc, sử dụng phương thức AddArc. Phương thức này sẽ tạo ra một cung tròn dựa trên 4 thông số khác nhau để xác định vị trí và kích thước của cung tròn:

**SetRetVal = object.AddArc(Center, Radius, StartAngle, EndAngle)**

Tham số	Giải thích
Center	Tham số đầu vào kiểu Variant hoặc mảng 3 phần tử kiểu Double, chứa tọa độ tâm của cung tròn.
Radius	Tham số đầu vào kiểu Double, là bán kính của đường tròn sẽ được tạo.
StartAngle, EndAngle	Tham số đầu vào kiểu Double, xác định góc bắt đầu và góc kết thúc của cung tròn (tính bằng Radian). Phương thức AddArc sẽ vẽ cung tròn theo chiều ngược chiều kim đồng hồ từ góc StartAngle đến góc EndAngle.
RetVal	Đối tượng kiểu Arc, tham chiếu đến cung tròn vừa mới được tạo.



**Hình V-9: Minh họa tham số StartAngle, EndAngle của phương thức AddArc.**

**CHÚ Ý** Hầu hết các tham số có liên quan đến góc trong VBA đều có đơn vị là Radian.

Đoạn mã sau tạo một cung tròn có tâm (0,0,0) và bán kính là 5 từ góc 45 đến 315 độ. Do giá trị tham số góc tính bằng radian nên cần phải chuyển đổi từ độ sang radian:

```
Sub Example_AddArc()
    Dim arcObj As AcadArc
    Dim centerPoint(0 To 2) As Double
    Dim radius As Double
    Dim startAngleInDegree As Double
    Dim endAngleInDegree As Double

    ' Xác định các thuộc tính của cung tròn
    centerPoint(0) = 0: centerPoint(1) = 0: centerPoint(2) = 0
    radius = 5                'Bán kính
    startAngleInDegree = 45   'Góc bắt đầu
    endAngleInDegree = 315   'Góc kết thúc

    ' Chuyển các góc từ độ sang Radian
    Dim startAngleInRadian As Double
    Dim endAngleInRadian As Double
    startAngleInRadian = startAngleInDegree * 3.141592 / 180
    endAngleInRadian = endAngleInDegree * 3.141592 / 180
End Sub
```

```
' Tạo đối tượng Arc trong không gian mô hình
Set arcObj = ThisDrawing.ModelSpace.AddArc _
    (centerPoint, radius, startAngleInRadian,
endAngleInRadian)
ZoomAll
End Sub
```

## 5.2.6. Tạo đối tượng văn bản

Văn bản là đối tượng dùng để truyền đạt những thông tin quan trọng trong bản vẽ. Ngoài ra, văn bản còn dùng để đặt tiêu đề cho khối, tạo nhãn cho từng thành phần của bản vẽ, thể hiện quy định chung hoặc để làm ghi chú trong bản vẽ.

AutoCAD cung cấp nhiều cách khác nhau để tạo văn bản, với những đoạn văn bản ngắn và đơn giản, có thể sử dụng văn bản đơn (Text), với những đoạn văn bản dài hơn, có chứa định dạng riêng bên trong thì có thể sử dụng văn bản nhiều dòng (MText). Mặc dù tất cả các đoạn văn bản mới được tạo đều sử dụng kiểu chữ hiện hành, với những thiết lập mặc định về phong chữ và định dạng nhưng cũng có nhiều cách khác nhau để tùy biến phần hiển thị của đoạn văn bản. Trong phạm vi của tài liệu này, chỉ giới thiệu về cách thức tạo đối tượng văn bản.

### Tạo văn bản đơn (Text)

Văn bản đơn là một đối tượng kiểu văn bản (Text) mà nội dung của nó chỉ bao gồm một dòng văn bản. Trong AutoCAD, để tạo đối tượng văn bản đơn trên bản vẽ, người dùng có thể sử dụng lệnh Text hoặc DText. Từ VBA, để tạo đối tượng văn bản đơn, sử dụng phương thức AddText của tập đối tượng ModelSpace, cú pháp của phương thức này như sau:

```
SetRetVal = object.AddText(TextString, InsertionPoint, Height)
```

Tham số	Giải thích
TextString	Kiểu String, là chuỗi sẽ được hiển thị trên bản vẽ.
InsertionPoint	Kiểu Variant (thực chất là mảng 3 phần tử kiểu Double) chứa tọa độ điểm bắt đầu chèn văn bản.
Height	Kiểu Double, xác định chiều cao của đoạn văn bản được hiển thị. Giá trị của tham số này phải là số dương. Nếu nhập vào giá trị $\leq 0$ , chương trình sẽ báo lỗi.
RetVal	Đối tượng kiểu Text, tham chiếu đến đối tượng văn bản đơn vừa mới được tạo.

Ví dụ sau sẽ tạo văn bản một dòng "Hello, World." trong không gian mô hình tại vị trí (2,2,0).

```
Sub VD_AddText()
    Dim textObj As AcadText
    Dim textString As String
    Dim insertionPoint(0 To 2) As Double
    Dim height As Double

    ' Tạo đối tượng Text
    textString = "Hello, World."
    insertionPoint(0) = 2
    insertionPoint(1) = 2
    insertionPoint(2) = 0
    height = 0.5
    Set textObj = ThisDrawing.ModelSpace._
        AddText(textString, insertionPoint, height)
    textObj.Update
End Sub
```

End Sub

### Tạo văn bản nhiều dòng (MText)

Đối với các đoạn văn bản dài và phức tạp, nên sử dụng đối tượng văn bản nhiều dòng – MText. Văn bản nhiều dòng được bố trí nằm trọn trong một bề rộng nhất định nhưng lại có thể mở rộng vô hạn theo chiều đứng. Ngoài ra, đối tượng MText còn có thể được định dạng chi tiết đến từng từ hoặc từng ký tự.

Mặc dù có nhiều dòng nhưng chúng thuộc về một đối tượng duy nhất. Đối tượng này có thể di chuyển, xoay, xóa, sao chép, lấy đối xứng, co giãn hoặc thay đổi tỷ lệ.

Để tạo đối tượng MText, sử dụng phương thức AddMText. Cú pháp của phương thức này như sau:

```
SetRetVal = object.AddMText(InsertionPoint, Width, Text)
```

Tham số	Giải thích
InsertionPoint	Kiểu Variant (thực chất là mảng 3 phần tử kiểu Double) chứa tọa độ điểm bắt đầu chèn văn bản.
Width	Kiểu Double, xác định chiều rộng của đoạn văn bản được hiển thị. Giá trị của tham số này phải là số dương. Nếu nhập vào giá trị $\leq 0$ , chương trình sẽ báo lỗi.
TextString	Kiểu String, là chuỗi sẽ được hiển thị trên bản vẽ.
RetVal	Đối tượng kiểu MText, tham chiếu đến đối tượng văn bản nhiều dòng vừa mới được tạo.

Thay vì sử dụng tham số Height như phương thức AddText, phương thức AddMText sử dụng tham số Width vì đối tượng MText có thể được hiển thị trên nhiều dòng.

Đoạn mã sau sẽ minh họa cách thức sử dụng phương thức AddMText:

```
Sub VD_AddMtext ()
    Dim MTextObj As AcadMText
    Dim corner(0 To 2) As Double
    Dim width As Double
    Dim text As String
    corner(0) = 0: corner(1) = 10: corner(2) = 0
    width = 5
    text = "Day la chuoi van ban cua doi tuong MText"
    ' Tạo đối tượng MText
    Set MTextObj = ThisDrawing.ModelSpace.AddMText(corner, width, text)
    ZoomAll
End Sub
```

### 5.3. Làm việc với đối tượng SelectionSet

Đối tượng SelectionSet thực chất là một tập đối tượng dùng để chứa các đối tượng được chọn trong bản vẽ (đối tượng hình học), tuy nhiên, để tránh nhầm lẫn, ta gọi nó là một đối tượng. Mỗi đối tượng SelectionSet đều có nhiều phương thức khác nhau dùng để thêm các đối tượng hình học vào trong nó. Thông thường, khi cần hiệu chỉnh chỉ với một đối tượng hình học duy nhất, ta chỉ cần sử dụng phương thức GetEntity của đối tượng Document.Utility (xem chi tiết cách sử dụng phương thức này trong mục **Error! Reference source not found.** trang **Error! Bookmark not defined.**) để lựa chọn đối tượng hình học cần hiệu chỉnh trên màn hình của

AutoCAD, nhưng để hiệu chỉnh một lúc nhiều đối tượng thì việc sử dụng đối tượng SelectionSet là thích hợp nhất.

Mỗi bản vẽ AutoCAD đều có một tập đối tượng tên là SelectionSets chứa tất cả các đối tượng SelectionSet trong bản vẽ. Người dùng có thể tạo ra nhiều đối tượng SelectionSet khác nhau bằng cách sử dụng phương thức Add có trong tập đối tượng SelectionSets. Một đặc điểm quan trọng của đối tượng SelectionSet chính là tính chất tạm thời của nó, sau khi đối tượng SelectionSet được tạo ra và người dùng đóng bản vẽ lại (có lưu những thay đổi) thì khi mở lại bản vẽ, tất cả các đối tượng SelectionSet đều đã bị xoá đi và chỉ còn lại tập đối tượng SelectionSets rỗng.

**CHÚ Ý** Tập đối tượng SelectionSets chứa các đối tượng SelectionSet trong bản vẽ. Các đối tượng SelectionSet sẽ bị xoá khi đóng bản vẽ. Vì vậy, lúc mới mở hoặc tạo bản vẽ, tập đối tượng SelectionSets luôn là tập rỗng.

Nhìn chung, quá trình làm việc với đối tượng SelectionSet cần phải trải qua các bước sau:

1. Khai báo đối tượng SelectionSet,
2. Khởi tạo đối tượng SelectionSet với lệnh Set của VB,
3. Thêm các đối tượng cần xử lý vào SelectionSet,
4. Thực hiện thao tác cần thiết trên các đối tượng trong SelectionSet.

### 5.3.1. Khai báo và khởi tạo đối tượng SelectionSet

Việc tạo đối tượng SelectionSet được thực hiện dễ dàng thông qua phương thức Add có trong tập đối tượng SelectionSets.

```
SetRetVal = object.Add(Name)
```

Tham số	Giải thích
Object	Là tập đối tượng SelectionSets
Name	Là chuỗi ký tự xác định tên của SelectionSet sẽ được tạo.
RetVal	Đối tượng SelectionSet tương ứng vừa mới được tạo ra.

Đoạn mã lệnh sau sẽ minh hoạ cách thức tạo đối tượng SelectionSet:

```
Sub VD_TaoSelectionSet()  
    Dim sset As AcadSelectionSet           `Khai báo biến  
    Set sset = ThisDrawing.SelectionSets.Add("MySSet") `Tạo  
    SelectionSet  
End Sub
```

Tuy nhiên, trong quá trình thao tác trên AutoCAD, khi sử dụng đối tượng SelectionSet, người dùng rất hay gặp lỗi với thông báo: đối tượng SelectionSet đã tồn tại. Chính vì vậy, để tránh lỗi này, nên sử dụng đoạn mã sau khi thực hiện tạo mới đối tượng SelectionSet:

```
1: Sub VD_GetXData()  
2:     Dim sset As AcadSelectionSet  
3:     On Error Resume Next  
4:     Set sset = ThisDrawing.SelectionSets("MySelectionSet")  
5:     If Err <> 0 Then  
6:         Err.Clear  
7:         Set sset = ThisDrawing.SelectionSets.Add("MySelectionSet")
```

```

8: Else
9:     sset.Clear
10: End If
11: End Sub
    
```

Dòng mã lệnh số 3 sẽ tắt thông báo lỗi. dòng mã lệnh số 4 sẽ thực hiện gán biến sset cho đối tượng SelectionSet có tên là MySelectionSet. Nếu đối tượng này chưa có thì đối tượng Err sẽ khác không ( $\neq 0$ ) và ta sẽ phải khởi tạo đối tượng SelectionSet này bằng phương thức Add ở dòng mã lệnh số 7, còn ngược lại, khi đối tượng SelectionSet có tên là MySelectionSet đã có trong tập đối tượng SelectionSets, để sử dụng nó, ta cần xóa bỏ nội dung mà nó đang chứa bên trong bằng dòng mã lệnh số 9.

### 5.3.2. Thêm đối tượng hình học vào một SelectionSet

Để thêm đối tượng hình học vào SelectionSet, người dùng có thể sử dụng các phương thức có sẵn trong đối tượng SelectionSet như AddItems hoặc họ phương thức SelectXXX, bao gồm: Select, SelectAtPoint, SelectOnScreen, SelectByPolygon. Phần dưới đây sẽ lần lượt giới thiệu về các phương thức trên.

#### Phương thức AddItems

Phương thức này dùng để thêm từng đối tượng vào trong SelectionSet. Cú pháp của phương thức này như sau:

```
object.AddItems Items
```

Tham số	Giải thích
Object	Là đối tượng SelectionSet.
Items	Kiểu Variant, là mảng chứa các đối tượng sẽ được thêm vào SelectionSet

Đoạn mã sau sẽ tạo một đối tượng SelectionSet có tên là “MySelectionSet”, sau đó tạo các đối tượng đường đa tuyến, đường thẳng, đường tròn và thêm các đối tượng này vào trong đối tượng SelectionSet.

```

Sub VD_AddItems ()

    Dim objs(0 To 2) As AcadEntity 'Mảng chứa các đối tượng mới được tạo

    ' Create the new selection set
    Dim ssetObj As AcadSelectionSet
    On Error Resume Next
    Set ssetObj = ThisDrawing.SelectionSets("MySelectionSet")
    If Err <> 0 Then
        Err.Clear
        Set ssetObj = ThisDrawing.SelectionSets.Add("MySelectionSet")
    Else
        ssetObj.Clear
    End If

    ' Tạo đường đa tuyến trong không gian mô hình
    Dim plineObj As AcadLWPolyline
    Dim points(0 To 5) As Double
    points(0) = 3: points(1) = 7
    points(2) = 9: points(3) = 2
    points(4) = 3: points(5) = 5
    
```

```

Set plineObj = ThisDrawing.ModelSpace.AddLightWeightPolyline(points)
plineObj.Closed = True
Set objs(0) = plineObj           `Thêm vào mảng các đối tượng

' Tạo đường thẳng trong không gian mô hình
Dim lineObj As AcadLine
Dim startPoint(0 To 2) As Double
Dim endPoint(0 To 2) As Double
startPoint(0) = 0: startPoint(1) = 0: startPoint(2) = 0
endPoint(0) = 2: endPoint(1) = 2: endPoint(2) = 0
Set lineObj = ThisDrawing.ModelSpace.AddLine(startPoint, endPoint)
Set objs(1) = lineObj           `Thêm vào mảng các đối tượng

' Tạo đường tròn trong không gian mô hình
Dim circObj As AcadCircle
Dim centerPt(0 To 2) As Double
Dim radius As Double
centerPt(0) = 20: centerPt(1) = 30: centerPt(2) = 0
radius = 3
Set circObj = ThisDrawing.ModelSpace.AddCircle(centerPt, radius)
Set objs(2) = circObj           `Thêm vào mảng các đối tượng
ZoomAll
    
```

```

' Thêm các đối tượng có trong mảng objs vào đối tượng SelectionSet
ssetObj.AddItem objs
ThisDrawing.Regen acAllViewports
End Sub
    
```

## Phương thức Select

Phương thức `Select` là phương thức cơ bản trong đối tượng `SelectionSet`. Với phương thức này, người dùng sẽ có nhiều lựa chọn khác nhau khi chọn đối tượng tùy thuộc vào các tham số của phương thức. Cú pháp của phương thức này như sau:

```
object.Select Mode[, Point1][, Point2][, FilterType][, FilterData]
```

Tham số	Giải thích
Object	Là đối tượng <code>SelectionSet</code>
Mode	Tham số xác định chế độ chọn đối tượng.
Point1	Tham số tùy chọn, kiểu <code>Variant</code> (mảng 3 phần tử kiểu <code>Double</code> ) chứa tọa độ điểm thứ nhất của cửa sổ lựa chọn, sử dụng kết hợp với <code>Point2</code>
Point2	Tham số tùy chọn, kiểu <code>Variant</code> (mảng 3 phần tử kiểu <code>Double</code> ) chứa tọa độ điểm thứ hai của cửa sổ lựa chọn, sử dụng kết hợp với <code>Point1</code>
FilterType, FilterData	Tham số tùy chọn, xác định bộ lọc đối tượng (Chi tiết tham khảo phần “Định nghĩa bộ lọc đối tượng cho <code>SelectionSet</code> ” trang 234).

Giá trị của tham số `Mode` sẽ xác định cách thức lựa chọn đối tượng khi sử dụng phương thức `Select`. Giá trị của tham số này có thể là một trong những giá trị sau:

Hằng số	Giá trị	Ý nghĩa
<code>acSelectionSetWindow</code>	0	Chọn tất cả các đối tượng nằm “trong” hình chữ nhật giới hạn bởi hai điểm <code>Point1</code> và <code>Point2</code>

acSelectionSetCrossing	1	Chọn tất cả các đối tượng nằm “trong” hoặc có giao với hình chữ nhật giới hạn bởi hai điểm Point1 và Point2
acSelectionSetPrevious	3	Chọn các đối tượng đã chọn gần nhất. Bỏ qua hai tham số Point1 và Point2.
acSelectionSetLast	4	Chọn đối tượng cuối cùng được tạo ra. Bỏ qua hai tham số Point1 và Point2.
acSelectionSetAll	5	Chọn tất cả các đối tượng đang có trong bản vẽ. Bỏ qua hai tham số Point1 và Point2.

Ví dụ sau sẽ minh họa cách sử dụng phương thức `Select` với tham số `Mode=acSelectionSetCrossing`:

```
Sub VD_Select()
    ' Tạo đối tượng SelectionSet
    Dim ssetObj As AcadSelectionSet
    On Error Resume Next
    Set ssetObj = ThisDrawing.SelectionSets("MySelectionSet")
    If Err <> 0 Then
        Err.Clear
        Set ssetObj = ThisDrawing.SelectionSets.Add("MySelectionSet")
    Else
        ssetObj.Clear
    End If

    ' Thêm tất cả các đối tượng nằm trong và giao với hình chữ nhật có
    ' tọa độ (28,17,0) và (-3.3, -3.6,0) vào trong đối tượng
    SelectionSet
    Dim mode As Integer
    Dim corner1(0 To 2) As Double
    Dim corner2(0 To 2) As Double
    mode = acSelectionSetCrossing
    corner1(0) = 28: corner1(1) = 17: corner1(2) = 0
    corner2(0) = -3.3: corner2(1) = -3.6: corner2(2) = 0
    ssetObj.Select mode, corner1, corner2
End Sub
```

### Phương thức `SelectAtPoint`

Phương thức này sẽ chọn các đối tượng đi qua một điểm cho trước để thêm vào `SelectionSet`. Cú pháp của phương thức này như sau:

```
object.SelectAtPoint Point [, FilterType] [, FilterData]
```

Tham số	Giải thích
Object	Là đối tượng <code>SelectionSet</code>
Point	Kiểu Variant (mảng 3 phần tử kiểu Double), chứa tọa độ điểm dùng để chọn đối tượng.
FilterType, FilterData	Tham số tùy chọn, xác định bộ lọc đối tượng (Chi tiết tham khảo phần “Định nghĩa bộ lọc đối tượng cho <code>SelectionSet</code> ” trang 234).

Ví dụ sau thêm tất cả các đối tượng đi qua điểm (6.8 , 9.4 , 0) vào đối tượng `SelectionSet` có tên là “MySelectionSet”:

```

Sub VD_SelectAtPoint()
  ' Tạo đối tượng SelectionSet
  Dim ssetObj As AcadSelectionSet
  On Error Resume Next
  Set ssetObj = ThisDrawing.SelectionSets("MySelectionSet")
  If Err <> 0 Then
    Err.Clear
    Set ssetObj = ThisDrawing.SelectionSets.Add("MySelectionSet")
  Else
    ssetObj.Clear
  End If

  ' Thêm tất cả các đối tượng qua điểm (6.8,9.4,0)
  ' vào đối tượng SelectionSet
  Dim point(0 To 2) As Double
  point(0) = 6.8: point(1) = 9.4: point(2) = 0
  ssetObj.SelectAtPoint point
End Sub
    
```

**Phương thức SelectByPolygon**

Phương thức này thực hiện chọn các đối tượng để thêm vào SelectionSet dựa trên mối tương quan với đường đa tuyến do người lập trình xác định trước. Cú pháp của phương thức này như sau:

```

object.SelectByPolygon Mode, PointsList[, FilterType][, FilterData]
    
```

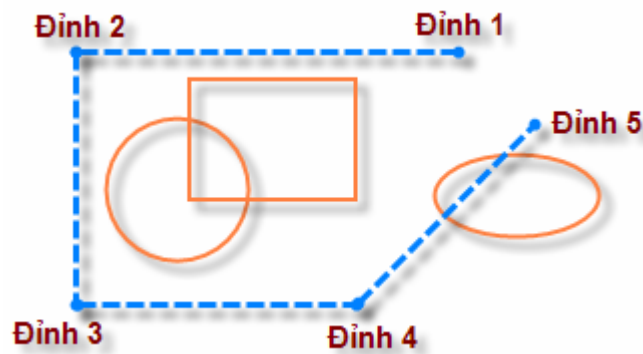
Tham số	Giải thích
Object	Là đối tượng SelectionSet
Mode	Tham số xác định chế độ chọn đối tượng.
PointsList	Tham số tùy chọn, kiểu Variant (mảng kiểu Double) chứa tọa độ 3 chiều của các đỉnh của đường đa tuyến.
FilterType, FilterData	Tham số tùy chọn, xác định bộ lọc đối tượng (Chi tiết tham khảo phần “Định nghĩa bộ lọc đối tượng cho SelectionSet” trang 234).

Giá trị của tham số Mode sẽ xác định cách thức lựa chọn đối tượng khi sử dụng phương thức SelectByPolygon. Giá trị của tham số này có thể là một trong những giá trị sau:

Hằng số	Giá trị	Ý nghĩa
acSelectionSetFence	2	Chọn các đối tượng có giao cắt với đường bao đa tuyến có tọa độ các đỉnh xác định bởi PointsList.
acSelectionSetWindowPolygon	6	Chọn các đối tượng nằm hoàn toàn bên trong miền đa giác có tọa độ các đỉnh xác định bởi PointsList.
acSelectionSetCrossingPolygon	7	Chọn các đối tượng nằm hoàn toàn hoặc một phần bên trong miền đa giác có tọa độ các đỉnh xác định bởi PointsList. AutoCAD sẽ tự động vẽ đa giác từ các tọa độ này theo nguyên tắc các cạnh của đa giác không giao nhau.

Minh họa dưới đây sẽ làm rõ ý nghĩa các giá trị của tham số Mode. Các đường liền là các đối tượng trên bản vẽ của AutoCAD, còn các đường nét đứt là đường đa tuyến nối các đỉnh được cho bởi tham số PointList.





Hình V-10: Minh họa các chế độ chọn đối tượng của phương thức `SelectByPolygon`.

Kết quả của phương thức `SelectByPolygon` là rất khác nhau tùy thuộc vào giá trị của tham số `Mode`. Nếu tham số `Mode` có giá trị là:

- ◆ `acSelectionSetFence`: đối tượng được chọn là hình ellipse.
- ◆ `acSelectionSetWindowPolygon`: đối tượng được chọn là hình chữ nhật và hình tròn.
- ◆ `acSelectionSetCrossingPolygon`: đối tượng được chọn là tất cả các hình trên.

Đoạn mã sau sẽ minh họa cách thức sử dụng phương thức `SelectByPolygon`

```
Sub VD_SelectByPolygon()
    ' Tạo đối tượng SelectionSet
    Dim ssetObj As AcadSelectionSet

    On Error Resume Next

    Set ssetObj = ThisDrawing.SelectionSets("MySelectionSet")
    If Err <> 0 Then
        Err.Clear
        Set ssetObj = ThisDrawing.SelectionSets.Add("MySelectionSet")
    Else
        ssetObj.Clear
    End If

    ' Xác định các đỉnh của đường đa tuyến
    Dim pointsArray(0 To 11) As Double
    pointsArray(0) = 28.2: pointsArray(1) = 17.2: pointsArray(2) = 0
    pointsArray(3) = -5: pointsArray(4) = 13: pointsArray(5) = 0
    pointsArray(6) = -3.3: pointsArray(7) = -3.6: pointsArray(8) = 0
    pointsArray(9) = 28: pointsArray(10) = -3: pointsArray(11) = 0
    ' Xác định chế độ chọn đối tượng
    Dim mode As Integer
    mode = acSelectionSetFence
    ' Chọn đối tượng
    ssetObj.SelectByPolygon mode, pointsArray
End Sub
```

### Phương thức `SelectOnScreen`

Phương thức này sẽ hiển thị dòng nhắc “Select object:” tại dòng lệnh và cho phép người dùng chọn đối tượng trực tiếp trên màn hình bản vẽ, cách thao tác trên màn hình AutoCAD này tương tự như khi sử dụng các lệnh thông thường khác của AutoCAD, mà có yêu cầu lựa chọn đối tượng (ví dụ như lệnh `Copy`). Cú pháp của phương thức này như sau:

```
object.SelectOnScreen [FilterType] [, FilterData]
```

Tham số	Giải thích
Object	Là đối tượng SelectionSet
FilterType, FilterData	Tham số tùy chọn, xác định bộ lọc đối tượng (Chi tiết tham khảo phần “Định nghĩa bộ lọc đối tượng cho SelectionSet” trang 234).

Đoạn mã sau sẽ minh họa cách thức sử dụng phương thức SelectOnScreen:

```
Sub VD_SelectOnScreen()
' Tạo đối tượng SelectionSet
Dim ssetObj As AcadSelectionSet
On Error Resume Next
Set ssetObj = ThisDrawing.SelectionSets("MySelectionSet")
If Err <> 0 Then
Err.Clear
Set ssetObj = ThisDrawing.SelectionSets.Add("MySelectionSet")
Else
ssetObj.Clear
End If

' Hiện thị thêm dòng nhắc tại dòng lệnh
ThisDrawing.Utility.Prompt vbCrLf & "Chon doi tuong tren man hinh:"
' Chọn đối tượng trên màn hình
ssetObj.SelectOnScreen
End Sub
```

### 5.3.3. Thao tác với các đối tượng trong SelectionSet

Như phần trước đã trình bày, thực chất đối tượng SelectionSet là một tập đối tượng dùng để nhóm các đối tượng hình học lại với nhau nhằm mục đích hiệu chỉnh các đối tượng hình học đó dễ dàng hơn. Do bản thân là một tập đối tượng nên cách thức truy cập đến tất cả các đối tượng bên trong SelectionSet tốt nhất là sử dụng cấu trúc lặp “For each ... next”.

Đoạn mã lệnh sau sẽ yêu cầu người sử dụng thực hiện chọn đối tượng trên màn hình bản vẽ, sau đó tiến hành đổi màu các đối tượng được chọn thành màu xanh.

```
Sub VD_SelectOnScreen()
' Tạo đối tượng SelectionSet
Dim ssetObj As AcadSelectionSet
On Error Resume Next
Set ssetObj = ThisDrawing.SelectionSets("MySelectionSet")
If Err <> 0 Then
Err.Clear
Set ssetObj = ThisDrawing.SelectionSets.Add("MySelectionSet")
Else
ssetObj.Clear
End If

' Chọn đối tượng trên màn hình
ThisDrawing.Utility.Prompt vbCrLf & "Chon doi tuong can doi mau:"
ssetObj.SelectOnScreen

' Thực hiện các thao tác với đối tượng được chọn
Dim ent As AcadEntity
For Each ent In ssetObj
' Đoạn chương trình xử lý các đối tượng sẽ nằm ở đây
' Trong ví dụ này, các đối tượng sẽ được đổi màu thành màu xanh
ent.Color = acBlue
ent.Update
End For
End Sub
```

```
Next ent
End Sub
```

### 5.3.4. Định nghĩa bộ lọc đối tượng cho SelectionSet

Trong tất cả các phương thức chọn đối tượng dạng `SelectXXX` đều có hai tham số tùy chọn `FilterType` và `FilterData`, là tham số được sử dụng để lọc các đối tượng được chọn theo một tiêu chí nào đó. Các tiêu chí thường được sử dụng bao gồm: loại đối tượng (đường thẳng, đường tròn...), màu sắc, kiểu đường nét, lớp... Khi sử dụng bộ lọc đối tượng, chỉ có những đối tượng thỏa mãn các tiêu chí trong bộ lọc mới được chọn để thêm vào đối tượng `SelectionSet`.

Để thiết lập bộ lọc đối tượng, hai tham số `FilterType` và `FilterData` cần phải được sử dụng song hành:

- ◆ `FilterType`: là tham số kiểu Variant, thực chất là một mảng một chiều kiểu Integer chứa mã DXF xác định kiểu lọc đối tượng.
- ◆ `FilterData`: là tham số kiểu Variant, thực chất là một mảng một chiều kiểu Variant chứa giá trị của kiểu lọc tương ứng trong tham số `FilterType`. Do có mối quan hệ 1-1 như vậy nên số phần tử của mảng `FilterData` phải bằng với số phần tử của mảng `FilterType`.

Tùy theo nhu cầu mà người lập trình phải chọn cho mình một tiêu chí lọc đối tượng thích hợp. Dưới đây sẽ liệt kê danh sách các mã DXF tương ứng với một số kiểu lọc đối tượng thường sử dụng:

Mã DXF	Ý nghĩa
-4	Các toán tử điều kiện (sử dụng cho bộ lọc theo nhiều điều kiện kết hợp)
0	Chuỗi thể hiện kiểu đối tượng, chẳng hạn như: Line, Polyline, LWPolyline, Spline, Circle, Arc, Text, Mtext,...
1	Chuỗi văn bản của các đối tượng như Text và MText (giá trị thuộc tính Content của các đối tượng này).
2	Chuỗi tương ứng với thuộc tính Name, chẳng hạn như thuộc tính Tag của đối tượng Attribute, thuộc tính name của đối tượng Block,...
6	Chuỗi tương ứng với kiểu đường (Linetype) của các đối tượng.
7	Chuỗi tương ứng với kiểu văn bản (Textstyle) của các đối tượng.
8	Chuỗi tương ứng với tên lớp (Layer) của các đối tượng.
10	Toạ độ điểm chủ yếu của đối tượng, chẳng hạn như: điểm đầu của đối tượng Line, điểm chèn của đối tượng Text, tâm của đối tượng Circle,...
62	Số nguyên xác định màu của đối tượng: 0-ByBlock, 256-ByLayer, 1-Red, 2-Yellow, ... các giá trị màu này tương ứng với bảng màu trong AutoCAD.

Tất cả các mã DXF có thể tham khảo trong tài liệu trợ giúp của AutoCAD “*DXF Reference*”, mục *DXF Format* ⇒ *Group Codes in Numerical Order*.

**CHÚ Ý** Khi truyền giá trị cho tham số `FilterType` và `FilterData`, số phần tử của mảng `FilterType` và mảng `FilterData` phải bằng nhau. Mỗi phần tử của mảng `FilterType` sẽ tương ứng với một phần tử của mảng `FilterData`.

### Lọc theo một điều kiện

Khi thực hiện lọc theo một điều kiện, số phần tử của các tham số `FilterType` và `FilterData` là 1. Tuy nhiên, người lập trình không được gán giá trị trực tiếp mà vẫn phải thực hiện khai báo các tham số này là mảng nhưng chỉ có một phần tử. Ví dụ sau sẽ minh họa rõ hơn cách thức tạo bộ lọc đối tượng với chỉ một điều kiện:

```

Sub VD_Filter()
  ' Tạo đối tượng SelectionSet
  Dim ssetObj As AcadSelectionSet
  On Error Resume Next
  Set ssetObj = ThisDrawing.SelectionSets("SSET")
  ssetObj.Delete
  Set ssetObj = ThisDrawing.SelectionSets.Add("SSET")

  ' Tạo bộ lọc đối tượng:
  ' Tiêu chí chọn là: Kiểu đối tượng
  ' Giá trị của tiêu chí là: "Circle"
  ' nghĩa là chỉ chọn đối tượng là đường tròn.
  Dim gpCode(0) As Integer
  Dim dataValue(0) As Variant
  gpCode(0) = 0: dataValue(0) = "Circle"

  ssetObj.SelectOnScreen gpCode, dataValue
  MsgBox "Số đối tượng được chọn: " & ssetObj.Count
End Sub

```

## Lọc theo nhiều điều kiện kết hợp

Khi cần lọc đối tượng theo nhiều điều kiện kết hợp, cần phải thêm vào các toán tử điều kiện trong bộ lọc. Để kết hợp các điều kiện với nhau, các điều kiện phải được đặt giữa các toán tử điều kiện, mã DXF chung của các toán tử điều kiện là -4. Dưới đây là danh sách các toán tử điều kiện được sử dụng trong bộ lọc đối tượng:

Toán tử	Giá trị bắt đầu	Giá trị kết thúc	Số lượng điều kiện	Ý nghĩa
AND	"<AND"	"AND>"	≥ 1	Đối tượng nào thoả mãn tất cả các điều kiện sẽ được chọn.
OR	"<OR"	"OR>"	≥ 1	Đối tượng nào thoả mãn một trong các điều kiện sẽ được chọn.
XOR	"<XOR"	"XOR>"	2	Đối tượng nào thoả mãn một điều kiện và không thoả mãn điều kiện còn lại sẽ được chọn.
NOT	"<NOT"	"NOT>"	1	Đối tượng nào không thoả mãn điều kiện sẽ được chọn.

Đoạn mã sau sẽ thực hiện chọn đối tượng có sử dụng bộ lọc theo nhiều điều kiện kết hợp: những đối tượng là đường thẳng hoặc đường tròn và không nằm trên lớp "Layer1" sẽ được chọn.

```

Sub VD_Filter()
  ' Tạo đối tượng SelectionSet
  Dim ssetObj As AcadSelectionSet
  On Error Resume Next
  Set ssetObj = ThisDrawing.SelectionSets("SSET")
  ssetObj.Delete
  Set ssetObj = ThisDrawing.SelectionSets.Add("SSET")

  ' Tạo bộ lọc đối tượng:
  Dim gpCode(8) As Integer
  Dim dataValue(8) As Variant
  gpCode(0) = -4: dataValue(0) = "<and"
  gpCode(1) = -4: dataValue(1) = "<or"

```

```

gpCode(2) = 0:      dataValue(2) = "line"
gpCode(3) = 0:      dataValue(3) = "circle"
gpCode(4) = -4:     dataValue(4) = "or>"

gpCode(5) = -4:     dataValue(5) = "<not"
gpCode(6) = 8:      dataValue(6) = "Layer1"
gpCode(7) = -4:     dataValue(7) = "not>"

gpCode(8) = -4:     dataValue(8) = "and>"

ssetObj.SelectOnScreen gpCode, dataValue
MsgBox "So doi tuong duoc chon: " & ssetObj.Count
End Sub
    
```

### 5.3.5. Loại bỏ đối tượng hình học ra khỏi SelectionSet

Khi muốn loại bỏ các đối tượng ra khỏi SelectionSet (tức là không muốn chọn đối tượng nữa), thì có thể sử dụng các phương thức sau của đối tượng SelectionSet.

#### Phương thức RemoveItems

Phương thức này thực hiện loại bỏ một hoặc nhiều đối tượng ra khỏi SelectionSet. Các đối tượng này sẽ không nằm trong SelectionSet nữa nhưng vẫn còn tồn tại trong bản vẽ. Cú pháp của phương thức này như sau:

```
object.RemoveItems Objects
```

Tham số	Giải thích
Object	Là đối tượng SelectionSet
Objects	Tham số kiểu Variant (mảng các đối tượng) chứa các đối tượng cần loại bỏ ra khỏi SelectionSet.

#### Phương thức Clear

Phương thức `Clear` sẽ loại bỏ tất cả các đối tượng ra khỏi SelectionSet. Sau khi thực hiện phương thức này, đối tượng SelectionSet vẫn còn tồn tại nhưng không chứa đối tượng nào cả. Các đối tượng hình học nằm trong SelectionSet lúc trước vẫn tồn tại trên bản vẽ nhưng không nằm trong đối tượng SelectionSet nữa. Cú pháp của phương thức này như sau:

```
object.Clear
```

Trong đó, object là đối tượng SelectionSet.

#### Phương thức Erase

Phương thức `Erase` không những loại bỏ tất cả các đối tượng hình học ra khỏi SelectionSet mà còn thực hiện xóa các đối tượng đó khỏi bản vẽ. Sau khi thực hiện phương thức này, đối tượng SelectionSet vẫn còn tồn tại nhưng không chứa đối tượng nào cả. Cú pháp của phương thức này như sau:

```
object.Erase
```

Trong đó, object là đối tượng SelectionSet.

## Phương thức Delete

Phương thức `Delete` sẽ xoá đối tượng `SelectionSet`. Sau khi thực hiện phương thức này, đối tượng `SelectionSet` sẽ không còn tồn tại trên bản vẽ, tuy nhiên các đối tượng hình học có trong `SelectionSet` lúc trước sẽ không bị xoá khỏi bản vẽ. Cú pháp của phương thức này như sau:

```
object.Delete
```

Trong đó, `object` là đối tượng `SelectionSet`.

### 5.4. Hiệu chỉnh đối tượng hình học

Hiệu chỉnh đối tượng hình học là một thao tác không thể thiếu trong quá trình tạo bản vẽ với AutoCAD. Thông qua VBA trong AutoCAD, người dùng có thể thực hiện hầu hết các thao tác hiệu chỉnh đối tượng giống như khi thực hiện trên giao diện của chương trình AutoCAD.

Việc hiệu chỉnh đối tượng có thể được thực hiện thông qua các phương thức hoặc các thuộc tính của đối tượng:

- ✦ Phương thức thường dùng để thay đổi về hình dạng, kích thước, vị trí của đối tượng, hoặc thậm chí có thể tạo mới đối tượng dựa trên đối tượng đã có. Thông thường, mỗi phương thức đều có những tham số riêng.
- ✦ Thuộc tính thường dùng để thay đổi các tính chất liên quan đến hiển thị của chính đối tượng đó trên màn hình hoặc khi in, chẳng hạn như màu sắc, kiểu đường, nét,...

Các đối tượng hình học trong AutoCAD, dù có khác nhau, nhưng đều được xây dựng dựa trên một giao tiếp cơ sở trong VBA: `IAcadEntity`. Chính vì vậy, tất cả các đối tượng hình học đều có những phương thức và thuộc tính chung. Bên cạnh đó, mỗi đối tượng này còn có những phương thức và thuộc tính riêng, chẳng hạn như phương thức `AddVertex` của đối tượng `LWPolyline`, hay thuộc tính `Radius` của đối tượng `Circle`,...

Do có những khác biệt như vậy nên trong phần này, trước hết sẽ giới thiệu cách thức để hiệu chỉnh chung cho tất cả các đối tượng hình học bao gồm:

- ✦ Sao chép, xoá, phá vỡ, di chuyển, lấy đối xứng, off-set, quay và co giãn đối tượng;
- ✦ Thực hiện nhân bản đối tượng dạng cục và dạng chữ nhật;
- ✦ Thao tác với dữ liệu mở rộng - `XData`;
- ✦ Thay đổi màu sắc, lớp, kiểu đường và sự hiển thị của đối tượng.

Và tiếp đó sẽ trình bày một số thao tác hiệu chỉnh đối tượng theo các phương thức và thuộc tính riêng của đối tượng:

- ✦ Hiệu chỉnh đường đa tuyến;
- ✦ Hiệu chỉnh văn bản đơn.

Trong các phiên bản trước AutoCAD 2006, mỗi khi hiệu chỉnh đối tượng bằng mã lệnh, những thay đổi đó sẽ chưa được hiển thị trên màn hình cho đến khi người dùng gọi phương thức `Update` của đối tượng đó, hoặc gọi phương thức `Update` của đối tượng `Application`, hoặc phương thức `Regen` của đối tượng `Document`. Trong một số trường hợp, AutoCAD sẽ tự động cập nhật khi kết thúc chương trình, tuy nhiên, để thấy được kết quả hiệu chỉnh ngay sau mỗi dòng mã lệnh hiệu chỉnh thì cách tốt nhất là nên bổ sung các đoạn mã lệnh cập nhật những thay đổi đó.

**CHÚ Ý** Để cập nhật những thay đổi đối với đối tượng thông qua mã lệnh, người dùng có thể sử dụng phương thức **Update** của chính đối tượng đó theo cấu trúc: **tên\_đối\_tượng.Update**.

### 5.4.1. Hiệu chỉnh đối tượng sử dụng các phương thức

#### Sao chép đối tượng – Phương thức Copy

Sử dụng phương thức `Copy` để sao chép đối tượng. Đối tượng mới được tạo ra sẽ giống hệt như đối tượng cũ, có vị trí trùng với đối tượng cũ, chỉ có điều là đối tượng mới sẽ được vẽ ở trên cùng. Cú pháp của phương thức này như sau:

```
SetRetVal = object.Copy
```

Tham số	Giải thích
Object	Đối tượng hình học, là đối tượng gốc sẽ được sao chép
RetVal	Đối tượng hình học, tham chiếu đến đối tượng vừa mới được tạo (là bản sao của đối <b>Object</b> )

**CHÚ Ý** Phương thức `Copy` chỉ sao chép đối tượng. Đối tượng mới được sao chép sẽ có vị trí trùng với đối tượng cũ.

Ví dụ sau tạo một hình tròn sau đó thực hiện sao chép hình tròn đó. Cần lưu ý là sau khi thực thi macro này, ta chỉ nhìn thấy trên màn hình bản vẽ một hình tròn nhưng thực chất là đã có hai hình tròn với vị trí trùng nhau.

```
Sub VD_Copy()
    ' Tạo hình tròn
    Dim circleObj As AcadCircle
    Dim center(0 To 2) As Double
    Dim radius As Double
    center(0) = 2#: center(1) = 2#: center(2) = 0#
    radius = 0.5
    Set circleObj = ThisDrawing.ModelSpace.AddCircle(center, radius)

    ' Sao chép đối tượng
    Dim copyCircleObj As AcadCircle
    Set copyCircleObj = circleObj.Copy()
End Sub
```

#### Xoá đối tượng – Phương thức Delete

Để xoá đối tượng khỏi bản vẽ, sử dụng phương thức `Delete` có trong đối tượng đó. Cú pháp của phương thức này rất đơn giản:

```
object.Delete
```

Ví dụ sau sẽ minh hoạ cách sử dụng phương thức này. Trong ví dụ này, người sử dụng sẽ chọn một đối tượng trên màn hình, và sau đó đối tượng này sẽ được xoá khỏi bản vẽ:

```
Sub DeleteObject()
    Dim objDrawingObject As AcadEntity
    Dim varEntityPickedPoint As Variant
```

```

On Error Resume Next
' Chọn đối tượng trên màn hình bản vẽ
ThisDrawing.Utility.GetEntity _
    objDrawingObject, varEntityPickedPoint, "Chọn đối tượng để xoá: "
If objDrawingObject Is Nothing Then
    MsgBox "Bạn chưa chọn đối tượng."
    Exit Sub
End If
' Xoá đối tượng được chọn
objDrawingObject.Delete
End Sub
    
```

### Phá vỡ đối tượng – Phương thức Explode

Sử dụng phương thức `Explode` để phá vỡ một đối tượng thành các đối tượng con. Phương thức này trả về một mảng tham chiếu đến các đối tượng con, là các đối tượng đã cấu thành nên đối tượng gốc. Cú pháp của phương thức này như sau:

```
SetRetVal = object.Explode
```

Tham số	Giải thích
Object	Đối tượng sẽ bị phá vỡ. Đối tượng này có thể là: 3DPolyline, BlockRef, LightweightPolyline, MInsertBlock, Polygonmesh, Polyline hoặc Region
RetVal	Mảng tham chiếu đến các đối tượng con cấu thành nên đối tượng <b>Object</b> .

Giá trị trả về của phương thức này là một mảng đối tượng với nhiều loại đối tượng khác nhau tùy thuộc vào loại đối tượng gốc. Chẳng hạn như khi phá vỡ một khối (đối tượng Block) thì kết quả trả về là các đối tượng cấu thành khối đó, hoặc khi phá vỡ một đường đa tuyến thì mảng đối tượng trả về là các đoạn thẳng của đường đa tuyến đó. Do giá trị trả về khác nhau như vậy nên khi khai báo mảng chứa giá trị trả về, nên sử dụng biến kiểu Variant.

Không giống như lệnh `Explode` trong AutoCAD, phương thức `Explode` không làm mất đối tượng gốc. Khi thực hiện phương thức này, một bản sao của đối tượng gốc sẽ được tạo ra, và sau đó, phương thức `Explode` mới thực hiện phá vỡ trên bản sao đó của đối tượng.

**CHÚ Ý** Phương thức `Explode` không thực hiện phá vỡ trên đối tượng gốc mà là trên bản sao của đối tượng gốc, vì vậy đối tượng gốc vẫn còn được giữ nguyên.

Ví dụ sau sẽ tạo ra một đường đa tuyến 2D và thực hiện phá vỡ đối tượng đó, sau đó sẽ duyệt qua các đối tượng thành phần sau khi đã được phá vỡ.

```

Sub VD_Explode()
    Dim plineObj As AcadLWPolyline
    Dim points(0 To 11) As Double

    ' Định nghĩa các điểm của đường đa tuyến
    points(0) = 1: points(1) = 1
    points(2) = 1: points(3) = 2
    points(4) = 2: points(5) = 2
    points(6) = 3: points(7) = 2
    points(8) = 4: points(9) = 4
    points(10) = 4: points(11) = 1

    ' Tạo đối tượng LWPolyline trong không gian mô hình
    Set plineObj = ThisDrawing.ModelSpace.AddLightWeightPolyline(points)
    
```



```

' Phá vỡ đường đa tuyến
MsgBox "Phá vỡ đường đa tuyến."
Dim explodedObjects As Variant
explodedObjects = plineObj.Explode

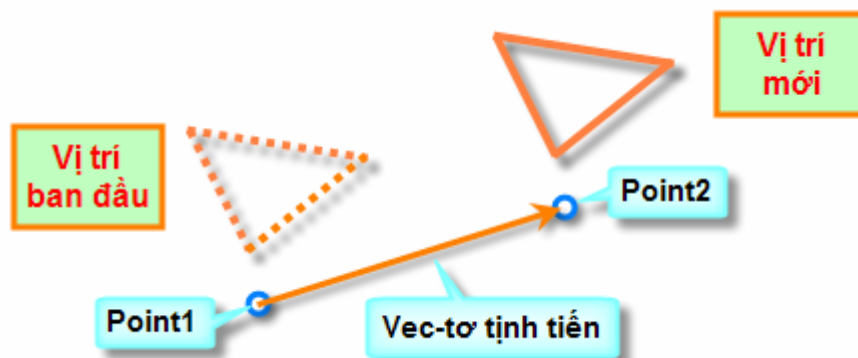
' Duyệt qua các đối tượng thành phần
Dim I As Integer
For I = 0 To UBound(explodedObjects)
    explodedObjects(I).Color = acRed
    explodedObjects(I).Update
    MsgBox "Đối tượng thứ " & I & ": " & explodedObjects(I).ObjectName
    explodedObjects(I).Color = acByLayer
    explodedObjects(I).Update
Next
End Sub
    
```

### Di chuyển đối tượng – Phương thức Move

Sử dụng phương thức `Move` để tịnh tiến đối tượng trong không gian ba chiều. Cú pháp của phương thức này như sau:

```
object.Move Point1, Point2
```

Tham số	Giải thích
Object	Đối tượng hình học, là đối tượng sẽ bị di chuyển.
Point1, Point2	Tham số đầu vào, là mảng 3 phần tử kiểu Double chứa tọa độ của điểm thứ nhất và thứ hai của vector tịnh tiến.



**Hình V-11: Minh họa phương thức Move**

Ví dụ sau sẽ minh họa cách thức sử dụng phương thức `Move` để dịch chuyển một đối tượng. Macro này sẽ tạo một đường tròn, sau đó dịch chuyển đường tròn này 2 đơn vị theo trục X.

```

Sub VD_Move()
' Tạo hình tròn trong không gian mô hình
Dim circleObj As AcadCircle
Dim center(0 To 2) As Double
Dim radius As Double
center(0) = 2#: center(1) = 2#: center(2) = 0#
radius = 0.5
Set circleObj = ThisDrawing.ModelSpace.AddCircle(center, radius)
ZoomAll
    
```

```
' Xác định 2 điểm tạo nên vec-tơ tịnh tiến
Dim point1(0 To 2) As Double
Dim point2(0 To 2) As Double
point1(0) = 0: point1(1) = 0: point1(2) = 0
point2(0) = 2: point2(1) = 0: point2(2) = 0

MsgBox "Di chuyển theo trục X 2 đơn vị."

' Thực hiện di chuyển đối tượng
circleObj.Move point1, point2

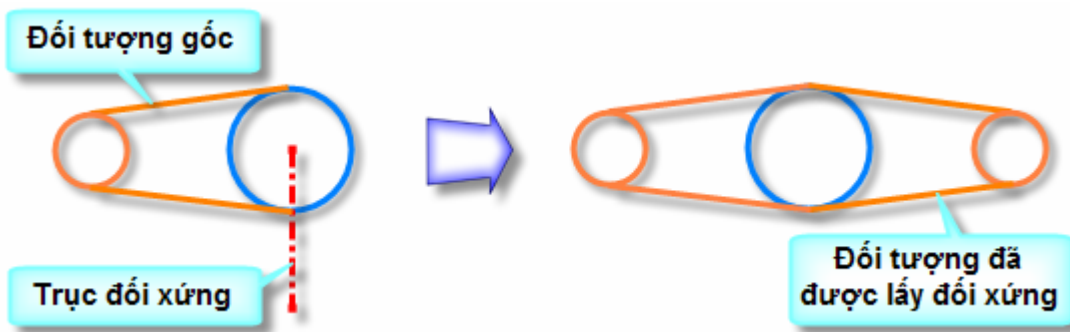
ZoomAll
MsgBox "Quá trình dịch chuyển đã kết thúc."
End Sub
```

## Lấy đối xứng – Phương thức Mirror

Sử dụng phương thức `Mirror` để lấy đối xứng một đối tượng qua một trục được xác định trước. Cú pháp của phương thức này như sau:

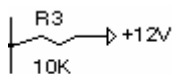
```
SetRetVal = object.Mirror(Point1, Point2)
```

Tham số	Giải thích
Object	Đối tượng hình học, là đối tượng gốc sẽ được lấy đối xứng.
Point1, Point2	Tham số đầu vào, là mảng 3 phần tử kiểu Double chứa tọa độ của điểm thứ nhất và thứ hai của trục đối xứng.
RetVal	Tham chiếu đến đối tượng sau khi đã được lấy đối xứng

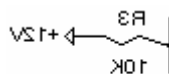


Minh họa: Phương thức Mirror.

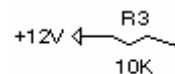
Khi sử dụng phương thức `Mirror`, cần lưu ý đến giá trị của biến hệ thống `MIRRTEXT`. Biến này sẽ điều khiển cách thức lấy đối xứng với đối tượng văn bản:



Trước khi lấy đối xứng



Sau khi lấy đối xứng  
(MIRRTEXT=1)



Sau khi lấy đối xứng  
(MIRRTEXT=0)

**Hình V-12: Biến hệ thống MIRRTEXT và phương thức Mirror**

Ví dụ sau sẽ tạo một đường đa tuyến, sau đó lấy đối xứng qua trục  $y=4.25$  và đổi màu đối tượng vừa mới được lấy đối xứng thành màu đỏ:

```

Sub VD_Mirror()

    ' Tạo đường đa tuyến
    Dim plineObj As AcadLWPolyline
    Dim points(0 To 11) As Double
    points(0) = 1: points(1) = 1
    points(2) = 1: points(3) = 2
    points(4) = 2: points(5) = 2
    points(6) = 3: points(7) = 2
    points(8) = 4: points(9) = 4
    points(10) = 4: points(11) = 1
    Set plineObj = ThisDrawing.ModelSpace.AddLightWeightPolyline(points)
    plineObj.Closed = True
    ZoomAll

    ' Xác định trục đối xứng
    Dim point1(0 To 2) As Double
    Dim point2(0 To 2) As Double
    point1(0) = 0: point1(1) = 4.25: point1(2) = 0
    point2(0) = 4: point2(1) = 4.25: point2(2) = 0

    MsgBox "Lấy đối xứng đường đa tuyến.", , "VD Mirror"

    ' Thực hiện lấy đối xứng đường đa tuyến
    Dim mirrorObj As AcadLWPolyline
    Set mirrorObj = plineObj.Mirror(point1, point2)
    mirrorObj.Color = acRed

    ZoomAll
    MsgBox "Mirror completed.", , " VD Mirror"
End Sub

```

### Lấy Off-set – Phương thức Offset

Phương thức Offset sẽ tạo ra một đối tượng mới với đường biên nằm cách đường biên của đối tượng cũ một khoảng được xác định trước. Phương thức này có thể được áp dụng với các đối tượng như: Arc, Circle, Ellipse, Line, LightweightPolyline, Polyline, Spline, và xline. Phương thức này sẽ trả về mảng chứa các đối tượng vừa mới được tạo. Tuy nhiên, thông thường mảng này chỉ chứa một đối tượng. Cú pháp của phương thức này như sau:

```
SetRetVal = object.Offset(Distance)
```

Tham số	Giải thích
Object	Là đối tượng gốc sẽ được lấy Off-set.
Distance	Tham số đầu vào, kiểu Double, là khoảng cách lấy off-set. Giá trị của tham số này có thể là số âm hoặc dương. Nếu là số âm thì phương thức này sẽ tạo ra những đối tượng có xu hướng “bán kính nhỏ hơn” đối tượng gốc, trong trường hợp “bán kính nhỏ hơn” không có ý nghĩa thì phương thức này sẽ tạo ra đối tượng có tọa độ X, Y và Z nhỏ hơn đối tượng gốc.
RetVal	Biến kiểu Variant, là mảng chứa các đối tượng mới được tạo ra. Thông thường, mảng này chỉ có một đối tượng.

Ví dụ sau sẽ tạo một đường đa tuyến trong không gian mô hình và lấy off-set đối tượng này một khoảng là 0,25. Đối tượng vừa mới được tạo ra sẽ được đổi màu thành màu đỏ.

```

Sub VD_Offset()
    ' Tạo đường đa tuyến
    Dim plineObj As AcadLWPolyline
    Dim points(0 To 11) As Double
    points(0) = 1: points(1) = 1
    points(2) = 1: points(3) = 2
    points(4) = 2: points(5) = 2
    points(6) = 3: points(7) = 2
    points(8) = 4: points(9) = 4
    points(10) = 4: points(11) = 1
    Set plineObj = ThisDrawing.ModelSpace.AddLightWeightPolyline(points)
    plineObj.Closed = True
    ZoomAll

    MsgBox "Off-set đa tuyến với khoảng cách 0.25.", , "Ví dụ Offset"

    ' Lấy Off-set đường đa tuyến
    Dim offsetObj As Variant
    offsetObj = plineObj.Offset(0.25)
    offsetObj(0).Color = acRed

    ZoomAll
End Sub

```

## Xoay đối tượng – Phương thức Rotate

Sử dụng phương thức `Rotate` để xoay một đối tượng quanh một điểm cố định. Cú pháp của phương thức này như sau:

```
object.Rotate BasePoint, RotationAngle
```

Tham số	Giải thích
Object	Là đối tượng sẽ được xoay.
BasePoint	Là mảng 3 phần tử kiểu Double chứa tọa độ điểm gốc, đối tượng sẽ được quay quanh điểm này.
RotationAngle	Là tham số kiểu Double, xác định góc xoay đối tượng (tính bằng Radian).

Đoạn mã sau sẽ minh họa cách thức sử dụng phương thức `Rotate`:

```

Sub VD_Rotate()
    ' Tạo đường đa tuyến
    Dim plineObj As AcadLWPolyline
    Dim points(0 To 11) As Double
    points(0) = 1: points(1) = 2
    points(2) = 1: points(3) = 3
    points(4) = 2: points(5) = 3
    points(6) = 3: points(7) = 3
    points(8) = 4: points(9) = 4
    points(10) = 4: points(11) = 2
    Set plineObj = ThisDrawing.ModelSpace.AddLightWeightPolyline(points)
    plineObj.Closed = True
    ZoomAll

    MsgBox "Xoay góc 45 độ.", , "VD Rotate "
    ' Định góc xoay và tọa độ điểm cơ sở
    Dim basePoint(0 To 2) As Double

```

```
Dim rotationAngle As Double
basePoint(0) = 4: basePoint(1) = 4.25: basePoint(2) = 0
rotationAngle = 45/180*3.1416      ' 45 độ

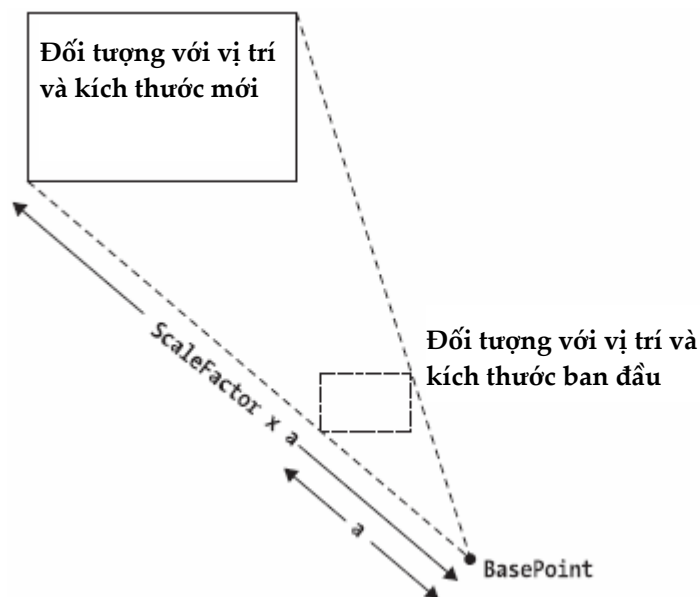
' Xoay đối tượng
plineObj.Rotate basePoint, rotationAngle
ZoomAll
End Sub
```

### Thay đổi tỷ lệ đối tượng – Phương thức ScaleEntity

Sử dụng phương thức `ScaleEntity` để thay đổi tỷ lệ đối tượng đồng đều theo các phương X, Y và Z. Cú pháp của phương thức này như sau:

**`object.ScaleEntity BasePoint, ScaleFactor`**

Tham số	Giải thích
Object	Là đối tượng sẽ được thay đổi tỷ lệ.
BasePoint	Mảng 3 phần tử kiểu Double chứa tọa độ điểm gốc, đối tượng sẽ được thay đổi tỷ lệ theo các phương X, Y và Z quanh điểm này quay quanh điểm này.
ScaleFactor	Tham số kiểu Double, xác định hệ số tỷ lệ khi thay đổi tỷ lệ. Kích thước của đối tượng sẽ được nhân với hệ số tỷ lệ này. Tham số <code>ScaleFactor</code> chỉ nhận giá trị lớn hơn không, nếu nhỏ hơn hoặc bằng không thì VBAIDE sẽ báo lỗi. Giá trị <code>ScaleFactor &lt; 1</code> sẽ thu nhỏ đối tượng, ngược lại sẽ phóng to đối tượng lên.



**Hình V-13: Minh họa phương thức ScaleEntity**

Đoạn mã sau tạo một đường đa tuyến trong không gian mô hình và sử dụng phương thức `ScaleEntity` để thay đổi tỷ lệ của đường đa tuyến đó quanh điểm (4 , 4.25 , 0) với hệ số tỷ lệ là 0.5

```
Sub VD_ScaleEntity()
' Tạo đường đa tuyến
Dim plineObj As AcadLWPolyline
Dim points(0 To 11) As Double
points(0) = 1: points(1) = 2
```

```

points(2) = 1: points(3) = 3
points(4) = 2: points(5) = 3
points(6) = 3: points(7) = 3
points(8) = 4: points(9) = 4
points(10) = 4: points(11) = 2
Set plineObj = ThisDrawing.ModelSpace.AddLightWeightPolyline(points)
plineObj.Closed = True
ZoomAll

MsgBox "Thay đổi tỷ lệ với hệ số 0.5", , "Ví dụ ScaleEntity"

' Xác định các tham số của phương thức
Dim basePoint(0 To 2) As Double
Dim scalefactor As Double
basePoint(0) = 4: basePoint(1) = 4.25: basePoint(2) = 0
scalefactor = 0.5

' Thay đổi tỷ lệ đường đa tuyến
plineObj.ScaleEntity basePoint, scalefactor
ZoomAll
End Sub
    
```

**5.4.2. Hiệu chỉnh đối tượng sử dụng các thuộc tính**

Không giống như các khi sử dụng các phương thức, người lập trình có thể sử dụng các thuộc tính để hiệu chỉnh sự hiển thị của các đối tượng hình học trong bản vẽ AutoCAD. Sau khi thay đổi thuộc tính của các đối tượng, nên sử dụng phương thức Update có trong đối tượng đó để cập nhật những thay đổi trên bản vẽ.

Dưới đây là các thuộc tính thường được sử dụng khi hiệu chỉnh các đối tượng hình học.

**Thuộc tính Color – Màu của đối tượng**

Sử dụng thuộc tính Color để lấy hoặc gán màu cho một đối tượng hình học. Thuộc tính này cũng có trong đối tượng Layer với ý nghĩa tương đương.

```
object.Color
```

Tham số	Giải thích
Object	Là đối tượng hình học, hoặc đối tượng Layer.
Color	Số nguyên hoặc hằng số acColor, xác định màu của đối tượng. Khi một đối tượng mới được tạo ra, giá trị mặc định của thuộc tính này acByLayer

Giá trị của thuộc tính Color là số nguyên trong phạm vi từ 0 đến 256, 9 trong các số này được định nghĩa trước với các hằng số acColor trong AutoCAD. Dưới đây là danh sách các hằng số này:

Hằng số	Giá trị	Ý nghĩa
acByBlock	0	Màu của đối tượng lấy theo màu của Block chứa đối tượng đó.
acRed	1	Màu đỏ.
acYellow	2	Màu vàng.
acGreen	3	Màu xanh lá.
acCyan	4	Màu xanh lam.

acBlue	5	Màu xanh da trời.
acMagenta	6	Màu tím
acWhite	7	Màu trắng hoặc đen tùy thuộc vào màu nền.
acByLayer	256	Màu của đối tượng lấy theo màu của lớp chứa đối tượng đó.

Đoạn mã dưới đây cho phép người dùng chọn đối tượng trên màn hình và thực hiện đổi màu đối tượng được chọn thành màu đỏ:

```
Sub VD_Color()
  Dim ent As AcadEntity
  Dim P(2) As Double
  ' Chọn đối tượng trên màn hình
  On Error Resume Next
  ThisDrawing.Utility.GetEntity ent, P, "Chon doi tuong can doi mau: "
  If Not (ent Is Nothing) Then
    ' Đổi màu đối tượng
    ent.Color = acRed
    ent.Update
  End If
End Sub
```

### Thuộc tính Layer – Lớp chứa đối tượng

Sử dụng thuộc tính `Layer` để lấy và thay đổi lớp chứa đối tượng. Thông thường, khi một đối tượng hình học mới được tạo ra, đối tượng đó sẽ được đặt trên lớp hiện hành của bản vẽ. Khi người dùng thay đổi giá trị của thuộc tính `Layer` thành tên của một lớp khác, đối tượng đó sẽ được tự động chuyển về nằm trên lớp mới này. Nếu người dùng gán cho thuộc tính `Layer` một tên lớp không có trong bản vẽ thì chương trình sẽ không báo lỗi, và đối tượng vẫn nằm trên lớp cũ.

#### `object.Layer`

Tham số	Giải thích
Object	Là đối tượng hình học.
Layer	Chuỗi chứa tên của lớp.

Đoạn mã dưới đây cho phép người dùng chọn đối tượng trên màn hình và thực hiện đổi lớp của đối tượng được chọn thành lớp "Layer1" (người dùng phải tạo trước một lớp có tên là "Layer1" bằng lệnh của AutoCAD):

```
Sub VD_Layer()
  Dim ent As AcadEntity
  Dim P(2) As Double
  ' Chọn đối tượng trên màn hình
  On Error Resume Next
  ThisDrawing.Utility.GetEntity ent, P, "Chon doi tuong can doi lop: "
  If Not (ent Is Nothing) Then
    ' Chuyển lớp cho đối tượng
    ent.Layer = "Layer1"
    ent.Update
  End If
End Sub
```

## Thuộc tính Linetype – Kiểu đường của đối tượng

Để thay đổi kiểu hiển thị của nét vẽ các đối tượng hình học trên màn hình, thiết lập lại thuộc tính `Linetype` của đối tượng thành một kiểu đường hiện có trong bản vẽ. Thông thường, khi một đối tượng vừa mới được tạo ra, đối tượng sẽ được gán kiểu đường bằng kiểu đường hiện hành trong bản vẽ. Thuộc tính này cũng có hiệu lực với đối tượng Layer.

**object.Linetype**

Tham số	Giải thích
Object	Là đối tượng hình học hoặc đối tượng Layer.
Linetype	Chuỗi chứa tên của kiểu đường của đối tượng. Ngoài ra, thuộc tính này cũng có thể bằng một trong những giá trị đặc biệt sau: CONTINUOUS: đây là kiểu đường mặc định, được AutoCAD tự động tạo ra. BYLAYER: kiểu đường của đối tượng sẽ được lấy bằng kiểu đường đã được gán cho lớp chứa đối tượng. BYBLOCK: kiểu đường của đối tượng sẽ được lấy bằng kiểu đường của block có chứa đối tượng.

Đoạn mã dưới đây cho phép người dùng chọn đối tượng trên màn hình và thực hiện đổi kiểu đường của đối tượng được chọn thành "DASHED2" (người dùng phải tạo trước kiểu đường có tên là "DASHED2" bằng lệnh của AutoCAD):

```
Sub VD_LineType()
    Dim ent As AcadEntity
    Dim P(2) As Double
    ' Chọn đối tượng trên màn hình
    On Error Resume Next
    ThisDrawing.Utility.GetEntity ent, P, "Chọn DT can doi kieu duong: "
    If Not (ent Is Nothing) Then
        ' Chuyển kiểu đường cho đối tượng
        ent.Linetype = "DASHED2"
        ent.Update
    End If
End Sub
```

**CHÚ Ý** có thể phải điều chỉnh giá trị của biến hệ thống LTSCALE thì mới quan sát được các kiểu đường không phải là nét liền.

## Thuộc tính Lineweight – Chiều dày nét in

Thuộc tính `Lineweight` dùng để thiết lập chiều dày nét in cho các đối tượng hình học và đối tượng Layer.

**object.Lineweight**

Giá trị của thuộc tính này là một hằng số kiểu `acLineWeight`:

Hằng số	Bề dày (mm)	Hằng số	Bề dày (mm)	Hằng số	Bề dày (mm)
<code>acLnWtByLayer</code>	Theo lớp	<code>acLnWt020</code>	0.2	<code>acLnWt080</code>	0.8



acLnWtByBlock	Theo Block	acLnWt025	0.25	acLnWt090	0.9
acLnWtByLwDefault	Mặc định	acLnWt030	0.3	acLnWt100	1.0
acLnWt000	0.0	acLnWt035	0.35	acLnWt106	1.06
acLnWt005	0.05	acLnWt040	0.4	acLnWt120	1.2
acLnWt009	0.09	acLnWt050	0.5	acLnWt140	1.4
acLnWt013	0.13	acLnWt053	0.53	acLnWt158	1.58
acLnWt015	0.15	acLnWt060	0.6	acLnWt200	2.0
acLnWt018	0.18	acLnWt070	0.7	acLnWt211	2.11

Khi một đối tượng hình học vừa mới được tạo ra, giá trị của thuộc tính này là `acLnWtByLayer`. Nếu không được gán bằng một chiều dày nét in cụ thể, chiều dày nét in sẽ được lấy bằng chiều dày mặc định của AutoCAD, chiều dày này được xác định thông qua biến hệ thống `LWDEFAULT` (giá trị này có đơn vị là 1/100 của mm).

Đoạn mã lệnh sau sẽ minh họa cách thức thao tác với thuộc tính `Lineweight`:

```
Sub VD_LineWeight ()
    Dim circleObj As AcadCircle
    Dim centerPoint(0 To 2) As Double
    Dim radius As Double

    ' Xác định các thông số của đường tròn

    centerPoint(0) = 0#: centerPoint(1) = 0#: centerPoint(2) = 0#
    radius = 5#
    ' Tạo đường tròn trong không gian mô hình
    Set circleObj = ThisDrawing.ModelSpace.AddCircle(centerPoint,
radius)
    ZoomAll
    ' Hiện thị chiều dày hiện hành của đường tròn
    MsgBox "Chiều dày hiện hành là: " & circleObj.Lineweight
    ' Thay đổi chiều dày của đường tròn
    circleObj.Lineweight = acLnWt211
    circleObj.Update
    MsgBox "Chiều dày hiện hành là: " & circleObj.Lineweight
End Sub
```

Ngoài những thuộc tính như đã giới thiệu ở trên, người lập trình có thể sử dụng các thuộc tính sau để hiệu chỉnh đối tượng:

Thuộc tính	Đối tượng áp dụng	Giải thích
Center	Arc, Circle.	Kiểu Variant (mảng 3 phần tử kiểu double). Toạ độ tâm của đối tượng.
Radius	Arc, Circle.	Kiểu Double. Bán kính của cung tròn hoặc đường tròn.
Area	Arc, Circle, LWPolyline, Polyline.	Kiểu Double. Diện tích vùng khép kín của đối tượng. là thuộc tính đọc-ghi đối với đối tượng Circle, là thuộc tính chỉ đọc với các đối tượng khác.
Length	Line	Kiểu Double. Thuộc tính chỉ đọc xác định chiều dài của đoạn thẳng.
TextString	MText, Text.	Kiểu String. Chuỗi văn bản được hiển thị trong đối tượng văn bản.

Alignment	Text.	Hằng số kiểu acAlignment, xác định chế độ canh hàng theo phương đứng và phương ngang.
Rotation	DimAligned, DimAngular, DimDiametric, DimOrdinate, DimRadial, DimRotated, MText, Text	Kiểu Double. Xác định góc xoay của đối tượng so với phương ngang, được tính bằng Radian.

### 5.4.3. Hiệu chỉnh đường đa tuyến

Như đã được giới thiệu ở phần trước, đối với đường đa tuyến 2D, có hai loại đối tượng là `LWPolyline` và `Polyline`, cách thức hiệu chỉnh hai đối tượng này là tương tự nhau. Tuy nhiên, cần phải chú ý một điểm khác biệt lớn giữa hai đối tượng này: toạ độ tại một đỉnh của đường `LWPolyline` chỉ có 2 thành phần X và Y, còn toạ độ tại một đỉnh của đường `Polyline` có cả 3 thành phần X, Y và Z nhưng thành phần thứ 3 sẽ không được sử dụng. Ở đây sẽ tập trung giới thiệu về cách hiệu chỉnh đường đa tuyến dạng `LWPolyline`.

Khi hiệu chỉnh đường đa tuyến, ngoài những cách thức đã được đề cập ở trên, người lập trình thường sử dụng các phương thức và thuộc tính sau:

Thuộc tính	Giải thích
Area	Double, chỉ đọc. Diện tích của vùng khép kín tạo bởi đường đa tuyến.
Closed	Boolean. Bằng TRUE: đường đa tuyến sẽ được khép kín, bằng FALSE: đường đa tuyến sẽ được mở.
ConstantWidth	Double. Chiều dày của tất cả các đoạn của đường đa tuyến.
Elevation	Double. Cao độ của đường đa tuyến (toạ độ z của đường đa tuyến).
Coordinate(i)	Variant (thực chất là mảng 2 hoặc 3 phần tử kiểu Double). Toạ độ của đỉnh thứ i của đường đa tuyến.
Coordinates	Variant (mảng kiểu Double). Toạ độ của tất cả các đỉnh của đường đa tuyến.

Phương thức	Giải thích
SetWidth	Thiết lập chiều dày đầu và chiều dày cuối cho đoạn thứ i của đường đa tuyến.
GetWidth	Lấy chiều dày đầu và chiều dày cuối cho đoạn thứ i của đường đa tuyến.

Dưới đây là một số thao tác hiệu chỉnh đường đa tuyến thường dùng

#### Thêm một đỉnh vào đường đa tuyến

Để thêm một đỉnh vào đường đa tuyến, có thể sử dụng phương thức `AddVertex`. Cú pháp của phương thức này như sau:

```
object.AddVertex Index, Point
```

Tham số	Giải thích
Object	Là đối tượng <code>LWPolyline</code> .
Index	Kiểu Integer. Chỉ số của đỉnh cần thêm vào đường đa tuyến. Đỉnh đầu tiên của đường đa tuyến được đánh số là 0. Giá trị Index nhập vào phải lớn hơn 0.
Point	Kiểu Variant (mảng 2 phần tử kiểu Double) chứa toạ độ của điểm cần thêm vào đường đa tuyến.

Ví dụ sau sẽ minh hoạ cách thức chèn một đỉnh vào đường đa tuyến:

```

Sub VD_AddVertex()
  Dim plineObj As AcadLWPolyline
  Dim points(0 To 9) As Double
  ' Xác định các đỉnh của đường đa tuyến
  points(0) = 1: points(1) = 1
  points(2) = 1: points(3) = 2
  points(4) = 2: points(5) = 2
  points(6) = 3: points(7) = 2
  points(8) = 4: points(9) = 4
  ' Tạo đường đa tuyến trong không gian mô hình
  Set plineObj = ThisDrawing.ModelSpace.AddLightWeightPolyline(points)
  ZoomAll
  MsgBox "Them dinh vao duong da tuyen.", , "Vi du AddVertex"

  ' Định nghĩa tọa độ đỉnh mới
  Dim newVertex(0 To 1) As Double
  newVertex(0) = 1.5: newVertex(1) = 1
  ' Thêm một đỉnh vào đường đa tuyến
  plineObj.AddVertex 2, newVertex
  plineObj.Update
  MsgBox "Da them dinh vao duong da tuyen.", , "Vi du AddVertex"
End Sub

```

### Đọc hoặc thay đổi tọa độ đỉnh của đường đa tuyến

Để đọc hoặc thay đổi tọa độ đỉnh của đường đa tuyến, có thể sử dụng thuộc tính `Coordinate(i)` có trong đối tượng `LWPolyline`. Khi sử dụng thuộc tính này, cần lưu ý là chỉ số các đỉnh của đường đa tuyến được đánh số bắt đầu từ 0.

Đoạn mã dưới đây sẽ tạo một đường đa tuyến, sau đó thay đổi tọa độ đỉnh thứ 2 của đường đa tuyến và hiển thị tọa độ mới của điểm này trong hộp thông báo.

```

Sub VD_Coordinate()
  Dim plineObj As AcadLWPolyline
  Dim points(0 To 9) As Double
  ' Xác định các đỉnh của đường đa tuyến
  points(0) = 1: points(1) = 1
  points(2) = 1: points(3) = 2
  points(4) = 2: points(5) = 2
  points(6) = 3: points(7) = 2
  points(8) = 4: points(9) = 4
  ' Tạo đường đa tuyến trong không gian mô hình
  Set plineObj = ThisDrawing.ModelSpace.AddLightWeightPolyline(points)
  ZoomAll
  MsgBox "Thay doi toa do dinh 2.", , "Vi du Coordinate"

  ' Thay đổi tọa độ của đỉnh thứ 2
  Dim newVertex(0 To 1) As Double
  newVertex(0) = 2: newVertex(1) = 1
  plineObj.Coordinate(2) = newVertex
  plineObj.Update

  ' Đọc lại tọa độ của đỉnh thứ 2
  Dim reaVertex As Variant
  reaVertex = plineObj.Coordinate(2)
  MsgBox "Toa do moi cua dinh 2: (" & _
    reaVertex(0) & "," & reaVertex(1) & ")", , "Vi du Coordinate"
End Sub

```

**5.4.4. Hiệu chỉnh văn bản đơn**

Đối tượng văn bản đơn (Text) được sử dụng rất nhiều trong các bản vẽ thiết kế, vì vậy, phần này chỉ tập trung giới thiệu cách thức hiệu chỉnh văn bản đơn thường dùng

**Thay đổi nội dung của văn bản**

Để thay đổi nội dung của văn bản đơn, sử dụng thuộc tính TextString có trong đối tượng Text. Ví dụ sau sẽ tạo một đối tượng văn bản đơn, sau đó thay đổi nội dung của văn bản đơn đó.

```
Sub Example_TextString()
    Dim textObj As AcadText
    Dim text As String
    Dim insPoint(0 To 2) As Double
    Dim height As Double
    ' Định nghĩa đối tượng văn bản đơn
    text = "Hello, World."
    insPoint(0) = 2: insPoint(1) = 2: insPoint(2) = 0
    height = 0.5

    ' Tạo văn bản đơn trong không gian mô hình
    Set textObj = ThisDrawing.ModelSpace.AddText(text, insPoint, height)
    ZoomAll

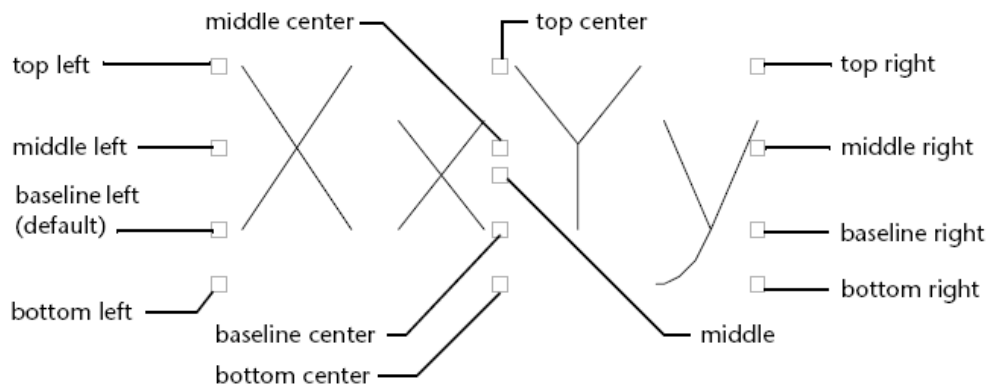
    ' Thay đổi giá trị của thuộc tính TextString
    MsgBox "Thuộc tính TextString là: " & textObj.TextString, _
        vbInformation, "Vi du TextString"
    textObj.TextString = "Chao cac ban!"
    textObj.Update
    MsgBox " TextString duoc thay doi thanh: " & textObj.TextString, _
        vbInformation, "Vi du TextString"
End Sub
```

**Thay đổi chế độ canh hàng**

Để thay đổi chế độ canh hàng của văn bản đơn, sử dụng thuộc tính Alignment. Giá trị của thuộc tính này có thể là một trong những hằng số acAlignment sau:

Hằng số	Hằng số	Hằng số
acAlignmentLeft	acAlignmentFit	acAlignmentMiddleCenter
acAlignmentCenter	acAlignmentTopLeft	acAlignmentMiddleRight
acAlignmentRight	acAlignmentTopCenter	acAlignmentBottomLeft
acAlignmentAligned	acAlignmentTopRight	acAlignmentBottomCenter
acAlignmentMiddle	acAlignmentMiddleLeft	acAlignmentBottomRight

Ý nghĩa của các hằng số này được minh họa trong hình sau:



Hình V-14: Minh họa các hằng số acAlignment.

Khi thực hiện thay đổi chế độ canh hàng của đối tượng, cần lưu ý đến tọa độ điểm được sử dụng để làm điểm canh hàng. Với từng chế độ canh hàng khác nhau, tọa độ điểm này có thể là tọa độ chứa trong thuộc tính InsertionPoint hoặc thuộc tính TextAlignmentPoint:

- ✦ Văn bản canh hàng bằng acAlignmentLeft sẽ sử dụng thuộc tính InsertionPoint để định vị trí của văn bản;
- ✦ Văn bản canh hàng bằng acAlignmentAligned hoặc acAlignmentFit sẽ sử dụng cả hai thuộc tính InsertionPoint và TextAlignmentPoint để định vị trí của văn bản;
- ✦ Văn bản canh hàng bằng các cách khác sẽ sử dụng thuộc tính TextAlignmentPoint để định vị trí văn bản.

Chính vì vậy, khi thay đổi chế độ canh hàng cho văn bản, cần phải xác định rõ chế độ canh hàng để xác định tọa độ điểm chèn của văn bản cho hợp lý.

Ví dụ sau sẽ minh họa cách thay đổi chế độ canh hàng cho văn bản. Trong ví dụ có tạo một điểm trong không gian mô hình dùng để tạo điểm tham chiếu để thấy rõ hơn sự thay đổi vị trí của văn bản khi thiết lập lại chế độ canh hàng

```

Sub VD_Alignment ()
    Dim textObj As AcadText

    Dim textString As String
    Dim insPoint(0 To 2) As Double, aliPoint(0 To 2) As Double
    Dim height As Double
    Dim oldPDMODE As Integer
    Dim pointObj As AcadPoint
    ' Định nghĩa đối tượng văn bản đơn
    textString = "Hello, World."
    insPoint(0) = 3: insPoint(1) = 3: insPoint(2) = 0
    aliPoint(0) = 3: aliPoint(1) = 3: aliPoint(2) = 0
    height = 0.5
    ' Tạo đối tượng văn bản trong không gian mô hình
    Set textObj = ThisDrawing.ModelSpace.AddText _
        (textString, insPoint, height)
    ' Lưu kiểu hiển thị điểm
    oldPDMODE = ThisDrawing.GetVariable("PDMODE")

    ' Tạo một điểm hình chữ thập tại điểm TextAlighmentPoint,
    ' để có thể hình dung rõ hơn sự thay đổi
    Set pointObj = ThisDrawing.ModelSpace.AddPoint(aliPoint)
    pointObj.Color = acRed
    ' Chuyển kiểu hiển thị điểm thành kiểu chữ thập
    ThisDrawing.SetVariable "PDMODE", 2
    ThisDrawing.Application.ZoomAll

```

```

' Thay chế độ canh hàng.
textObj.Alignment = acAlignmentRight
' Do chế độ canh hàng mới là acAlignmentRight nên điểm tham chiếu
' để định vị văn bản là TextAlignmentPoint
' Vì vậy, cần phải gán toạ độ cho điểm TextAlignmentPoint
textObj.TextAlignmentPoint = aliPoint
textObj.Update
MsgBox "Van ban dang o che do canh phai"

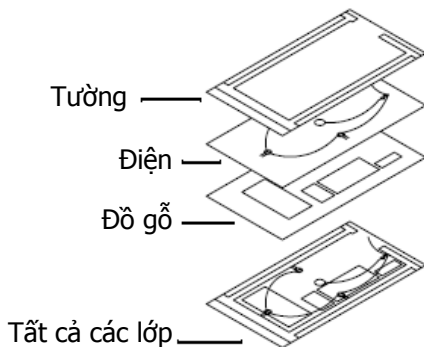
' Canh giữa văn bản với điểm tham chiếu là điểm TextAlignmentPoint
textObj.Alignment = acAlignmentCenter
textObj.Update
MsgBox "Van ban dang o che do canh giua"

' Hoàn trả lại kiểu hiển thị điểm
ThisDrawing.SetVariable "PDMODE", oldPDMODE
End Sub
    
```

## 5.5. Làm việc với lớp (Layer)

Lớp là các tầng trong suốt mà trên đó ta có thể tổ chức và nhóm nhiều loại đối tượng hình học khác nhau của bản vẽ. Việc sắp xếp các lớp và đối tượng trên lớp giúp quản lý thông tin của bản vẽ dễ dàng hơn.

Quá trình vẽ luôn được thực hiện trên một lớp nào đó. Đó có thể là lớp mặc định hoặc một lớp tự tạo ra. Mỗi lớp đều có một màu và kiểu đường tương ứng. Chẳng hạn, người lập trình có thể tạo ra một lớp mà trên đó chỉ vẽ các đường trục và gán cho nó màu xanh với kiểu đường là CENTER. Tiếp đó, cứ khi nào muốn vẽ các đường trục ta chỉ cần chuyển sang lớp đó và bắt đầu vẽ. Không nhất thiết là phải thiết lập kiểu đường và màu sắc mỗi khi muốn vẽ một đường trục. Ngoài ra, cũng có thể tắt lớp đường trục nếu không muốn hiển thị hoặc in các đối tượng này. Sử dụng lớp là một trong những ưu điểm lớn khi vẽ bằng AutoCAD so với khi sử dụng với giấy bút thông thường.



### 5.5.1. Tạo lớp mới

Người lập trình có thể tạo và đặt tên lớp cho một nhóm các đối tượng có ý nghĩa chung nào đó (chẳng hạn như lớp các bức tường hoặc lớp kích thước) và gán màu và kiểu đường cho những lớp này. Khi sắp xếp sơ đồ lớp, nên chọn tên lớp sao cho có ý nghĩa.

Khi bắt đầu một bản vẽ mới, AutoCAD sẽ tạo ra một lớp đặc biệt có tên là 0. Lớp 0 này được mặc định gán cho màu số 7 (màu trắng hoặc màu đen tùy thuộc vào giá trị màu nền) và với kiểu đường CONTINUOUS (liên tục). Và người dùng không thể xoá lớp 0 này.

Người lập trình có thể tạo ra lớp mới và gán thuộc tính màu sắc cũng như kiểu đường cho những lớp này. Mỗi một lớp là một phần tử của tập đối tượng Layers. Và để tạo một lớp mới, có thể sử dụng phương thức Add có trong tập đối tượng Layers. Cú pháp của phương thức này như sau:

```
Set RetVal = object.Add(Name)
```

Tham số	Giải thích
Object	Là tập đối tượng Layers.
Name	Kiểu String. Xác định tên của lớp sẽ được tạo trong bản vẽ.
RetVal	Kiểu Layer. Biến tham chiếu đến đối tượng Layer vừa mới được tạo.

Đoạn mã lệnh sau sẽ tạo lớp mới tên “New\_Layer”, gán màu cho lớp và sau đó hiển thị các thông tin của lớp trong một hộp thông báo.

```
Sub VD_AddLayer()
    Dim layerObj As AcadLayer
    ' Thêm lớp mới vào tập đối tượng Layers
    Set layerObj = ThisDrawing.Layers.Add("New_Layer")
    ' Make the new layer the active layer for the drawing
    layerObj.Color = acRed
    ' Display the status of the new layer
    MsgBox layerObj.Name & " has been added." & vbCrLf & _
        "LayerOn Status: " & layerObj.LayerOn & vbCrLf & _
        "Freeze Status: " & layerObj.Freeze & vbCrLf & _
        "Lock Status: " & layerObj.Lock & vbCrLf & _
        "Color: " & layerObj.Color, , "Add Example"
End Sub
```

**CHÚ Ý** Khi tạo một lớp mới, nếu tham số Name trùng tên với một lớp đã có thì lớp đã có sẽ không bị xoá đi và chương trình sẽ không báo lỗi. Khi đó, đối tượng RetVal sẽ được tự động tham chiếu đến lớp có tên lớp bằng với giá trị của tham số Name.

### 5.5.2. Truy xuất và thay đổi tên một lớp đã có

Khi cần thực hiện một tác động nào đó đến một lớp, người lập trình cần phải thực hiện thao tác truy cập đến lớp đó. Để làm được việc này, người lập trình phải thực hiện thông qua tập đối tượng Layers có trong đối tượng Document, vì tất cả các lớp đều được chứa trong tập đối tượng này. Cách thức truy xuất đến một lớp có tên là “New\_Layer” đã có trong bản vẽ như sau:

```
Dim layerObj As AcadLayer
Set layerObj = ThisDrawing.Layers ("New_Layer")
```

Ngoài ra, người lập trình cũng có thể sử dụng vòng lặp For Each để duyệt qua tất cả các lớp đang có trong bản vẽ.

Đoạn mã dưới đây sẽ duyệt qua tất cả các lớp trong tập đối tượng Layers, hiển thị tên của các lớp trong hộp thông báo và nếu lớp nào có tên là “New\_Layer” sẽ đổi tên thành “Tuong\_Canh”:

```
Sub VD_DuyetLop()
    Dim layerNames As String
    Dim ent As AcadLayer
    layerNames = ""
    For Each ent In ThisDrawing.Layers
        If ent.Name = "New_Layer" Then
            ent.Name = "Tuong_Canh"
        End If
    End For
End Sub
```

```

    layerNames = layerNames + ent.Name + vbCrLf
Next
MsgBox "Danh sach cac lop: " + vbCrLf + layerNames
End Sub

```

### 5.5.3. Thiết lập lớp hiện hành

Quá trình vẽ luôn được thực hiện trên lớp hiện hành. Sau khi chọn lớp thành lớp hiện hành, các đối tượng hình học mới được tạo ra đều được đặt trên lớp đó và tất cả các đối tượng này đều sử dụng màu và kiểu đường của lớp đó. Cần lưu ý là: không thể thiết lập thành lớp hiện hành cho một lớp nào đó nếu như lớp đó đã đông cứng (Freeze).

Để chọn một lớp làm lớp hiện hành, ta sử dụng thuộc tính `ActiveLayer` trong đối tượng `Document`. Đoạn mã dưới đây sẽ tạo một lớp mới và biến lớp đó thành lớp hiện hành:

```

Dim newlayer As AcadLayer
Set newlayer = ThisDrawing.Layers.Add("LAYER1")
ThisDrawing.ActiveLayer = newlayer

```

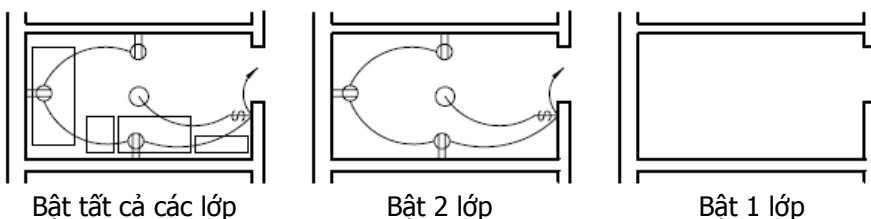
### 5.5.4. Thiết lập các chế độ hiển thị của lớp

AutoCAD không hiển thị hay in các đối tượng được vẽ trên các lớp ẩn. Nếu muốn không bị vướng tầm nhìn khi làm việc chi tiết trên một lớp hoặc một số lớp cụ thể, hoặc nếu không muốn in một số chi tiết nhất định, chẳng hạn như đường chuẩn hoặc các đường tạm, ta có thể tắt hoặc làm đông cứng các lớp không cần thiết đó.

Hình thức điều khiển tính nhìn thấy của một lớp phụ thuộc vào cách ta làm việc và kích thước bản vẽ, bao gồm: bật/tắt lớp, khoá/mở khoá lớp, đông cứng/tan lớp.

#### Bật/Tắt lớp

Các lớp đã tắt sẽ vẫn được tái tạo lại cùng với bản vẽ nhưng lại không được hiển thị hay in ra. Nên tắt các lớp đi thay vì làm đông cứng nếu thường xuyên chuyển đổi giữa các lớp có nhìn thấy và lớp ẩn. Bằng cách tắt các lớp, ta tránh phải tái tạo lại bản vẽ mỗi khi làm tan lớp. Khi bật một lớp đã được tắt, AutoCAD sẽ vẽ lại các đối tượng trên lớp đó.



Để bật hay tắt các lớp, ta sử dụng thuộc tính `LayerOn`. Để bật một lớp, gán giá trị `TRUE` và để tắt một lớp thì gán giá trị `FALSE` cho thuộc tính này.

Trong ví dụ sau, ta tạo một lớp mới, tiếp đó thêm một vòng tròn và tắt lớp đó đi để không nhìn thấy vòng tròn nữa.

```

Sub VD_LayerInvisble()
' Tạo đường tròn
Dim circleObj As AcadCircle
Dim center(0 To 2) As Double
Dim radius As Double
center(0) = 2: center(1) = 2: center(2) = 0
radius = 1
Set circleObj = ThisDrawing.ModelSpace. _
AddCircle(center, radius)

```



```

circleObj.Color = acByLayer
' Tạo lớp "ABC"
Dim layerObj As AcadLayer
Set layerObj = ThisDrawing.Layers.Add("ABC")
layerObj.Color = acRed
' Gán đường tròn vào lớp "ABC"
circleObj.Layer = "ABC"
circleObj.Update
' Tắt lớp "ABC"
layerObj.LayerOn = False
ThisDrawing.Regen acActiveViewport
End Sub

```

### Đông cứng/Tan lớp

Người lập trình có thể làm đông cứng các lớp không cần hiển thị trong thời gian dài để tăng tốc độ hiển thị những thay đổi, cải thiện quá trình lựa chọn đối tượng và giảm thời gian tái tạo cho những bản vẽ phức tạp. AutoCAD không hiển thị, in hay tái tạo các đối tượng trên các lớp đông cứng. Khi ta làm “tan” một lớp bị đông cứng, AutoCAD sẽ phục hồi và hiển thị các đối tượng trên lớp đó.

Để làm đông cứng hay làm tan một lớp bị đông cứng, ta sử dụng thuộc tính `Freeze`. Để làm đông cứng một lớp, ta gán giá trị `TRUE` và để làm tan một lớp, ta gán giá trị `FALSE` cho thuộc tính này.

Ví dụ sau tạo một lớp mới có tên “ABC” và sau đó làm đông cứng lớp đó.

```

Sub VD_LayerFreeze()
' Tạo lớp "ABC"
Dim layerObj As AcadLayer
Set layerObj = ThisDrawing.Layers.Add("ABC")
' Đông cứng lớp "ABC"
layerObj.Freeze = True
End Sub

```

### Khoá/Mở khoá lớp

Khi một lớp đã bị khoá, các đối tượng nằm trên lớp đó sẽ không thể hiệu chỉnh được nữa mặc dù người dùng vẫn nhìn thấy chúng trên bản vẽ. Biện pháp khoá lớp rất hữu ích khi người dùng muốn hiệu chỉnh các đối tượng trên một lớp nào đó nhưng đồng thời cũng muốn quan sát các đối tượng ở các lớp khác mà không làm ảnh hưởng đến các đối tượng ở lớp này. Người dùng có thể biến một lớp đã bị khoá thành lớp hiện hành, thêm đối tượng vào lớp, màu và kiểu đường của các lớp,... nhưng không thể hiệu chỉnh các đối tượng đã có trong lớp.

Để khoá hay mở các lớp, ta sử dụng thuộc tính `Lock`. Nếu muốn khoá một lớp, ta gán giá trị `TRUE`, mở khoá ta gán giá trị `FALSE` cho thuộc tính này.

Ví dụ sau tạo một lớp có tên “ABC” và sau đó khoá lớp này lại.

```

Sub VD_LayerLock()
' Tạo lớp "ABC"
Dim layerObj As AcadLayer
Set layerObj = ThisDrawing.Layers.Add("ABC")
' Khóa lớp "ABC"
layerObj.Lock = True
End Sub

```

#### 5.5.5. Xoá lớp

Để xóa lớp, sử dụng phương thức `Delete` của chính đối tượng `Layer`.

Lớp có thể được xóa bất kỳ lúc nào trong khi vẽ. Tuy nhiên, không thể xóa lớp hiện hành, lớp 0, hoặc lớp đang có chứa đối tượng, nếu cố tình thực hiện, chương trình sẽ phát sinh lỗi.

**GỢI Ý** Để xóa các đối tượng phi hình học (bao gồm cả đối tượng `Layer`) không còn được sử dụng trong bản vẽ, có thể sử dụng phương thức `PurgeAll` có trong đối tượng `Document`, hoặc sử dụng lệnh `Purge` tại dòng lệnh của `AutoCAD`.

Đoạn mã sau sẽ thực hiện xóa lớp có tên là "ABC".

```
Sub VD_LayerDelete()  
    Dim layerObj As AcadLayer  
    Set layerObj = ThisDrawing.Layers.Add("ABC")  
    layerObj.Delete  
End Sub
```

### 5.6. Thao tác với kiểu đường – Linetype

Kiểu đường là sự lặp lại mẫu gạch ngang, chấm, các khoảng trống hoặc các ký hiệu khác. Trong `AutoCAD`, các kiểu đường được định nghĩa sẵn trong tệp `ACAD.LIN`. Khi muốn sử dụng một kiểu đường nào đó, cần phải thực hiện tải kiểu đường đó vào trong `AutoCAD`.

**GỢI Ý** Đối với `AutoCAD 2002`, thông thường, tệp `ACAD.LIN` sẽ được lưu trong thư mục "C:\Program Files\AutoCAD 2002\Support"

#### 5.6.1. Tải kiểu đường vào AutoCAD

Để tải kiểu đường vào `AutoCAD`, sử dụng phương thức `Load` có trong tập đối tượng `Linetypes`. Những kiểu đường đã được tải vào đều nằm trong tập đối tượng `Linetypes`. Cú pháp của phương thức `Load` như sau:

```
object.Load LineTypeName, FileName
```

Tham số	Giải thích
Object	Là tập đối tượng <code>Linetypes</code>
LineTypeName	Kiểu String. Tên của kiểu đường cần tải vào <code>AutoCAD</code> (tên của kiểu đường phải có trong tệp chứa kiểu đường).
FileName	Kiểu String. Tên tệp chứa kiểu đường cần tải vào <code>AutoCAD</code> .

Ví dụ sau thực hiện tải kiểu đường "CENTER" từ tệp `ACAD.LIN`. Nếu kiểu đường này đã có, thì một thông báo sẽ xuất hiện để báo cho người dùng được biết:

```
Sub VD_LinetypeLoad()  
    Dim linetypeName As String  
    linetypeName = "CENTER"  
  
    ' Tải kiểu đường "CENTER" từ tệp acad.lin  
    On Error Resume Next  
    ThisDrawing.Linetypes.Load linetypeName, "acad.lin" ' bẫy lỗi  
  
    ' Nếu kiểu đường đã tồn tại, thông báo cho người dùng  
    If Err.Description = "Duplicate record name" Then  
        MsgBox "Kieu duong ten '" & linetypeName & "' da duoc tai."  
    End If
```

End Sub

### 5.6.2. Truy xuất và đổi tên kiểu đường

Khi cần thực hiện một tác động nào đó đến kiểu đường, người lập trình cần phải thực hiện thao tác truy xuất đến kiểu đường đó. Để làm được việc này, người lập trình phải thực hiện thông qua tập đối tượng Linetypes có trong đối tượng Document vì tất cả các kiểu đường đều được chứa trong tập đối tượng này. Cách thức truy xuất đến kiểu đường có tên là "CENTER" đã có trong bản vẽ như sau:

```
Dim linetypeObj As AcadLineType
Set linetypeObj = ThisDrawing.Linetypes("CENTER")
```

Người dùng có thể đổi tên một kiểu đường để dễ nhận biết hơn trong quá trình sử dụng. Khi đổi tên kiểu đường, thực chất là chỉ đổi tên kiểu đường định nghĩa trong bản vẽ. Tên kiểu đường trong tệp .LIN vẫn giữ nguyên. Để đổi tên kiểu đường, ta sử dụng thuộc tính Name có trong đối tượng Linetype.

**CHÚ Ý** Không thể đổi tên kiểu đường BYLAYER, BYBLOCK, hoặc CONTINUOUS.

Ví dụ sau thực hiện việc truy xuất đến kiểu đường tên là CENTER và đổi tên kiểu đường thành "Duong\_Tam". Nếu kiểu đường này chưa có trong bản vẽ, sẽ thông báo lỗi cho người dùng:

```
Sub VD_LinetypeName()
    Dim linetypeObj As AcadLineType
    On Error Resume Next
    ' Thực hiện truy xuất đến kiểu đường
    Set linetypeObj = ThisDrawing.Linetypes("CENTER")
    If linetypeObj Is Nothing Then
        ' Thông báo lỗi khi kiểu đường không tồn tại
        MsgBox "Kieu duong khong ton tai"
    Else
        ' Thay đổi tên kiểu đường
        linetypeObj.Name = "Duong_Tam"
    End If
End Sub
```

### 5.6.3. Thiết lập kiểu đường hiện hành

Khi một kiểu đường được kích hoạt làm kiểu đường hiện hành, tất cả các đối tượng mới được tạo sẽ sử dụng kiểu đường này, trừ khi đối tượng đó có kiểu đường là BYLAYER hoặc BYBLOCK. Để kích hoạt một kiểu đường thành kiểu đường hiện hành, sử dụng thuộc tính ActiveLinetype có trong đối tượng Document.

Đoạn mã sau sẽ kích hoạt kiểu đường "CENTER" thành kiểu đường hiện hành của bản vẽ:

```
Sub VD_ActiveLinetype()
    Dim linetypeObj As AcadLineType
    Set linetypeObj = ThisDrawing.Linetypes("CENTER")
    ThisDrawing.ActiveLinetype = linetypeObj
End Sub
```

### 5.6.4. Xoá kiểu đường đã có

Khi không cần sử dụng một kiểu đường nào đó, có thể xóa kiểu đường ra khỏi bản vẽ. Để thực hiện thao tác này, sử dụng phương thức Delete có trong đối tượng Linetype.

Tuy nhiên không thể xóa kiểu đường “BYLAYER”, “BYBLOCK”, “CONTINUOUS”, kiểu đường hiện hành hoặc kiểu đường đang được một đối tượng khác sử dụng.

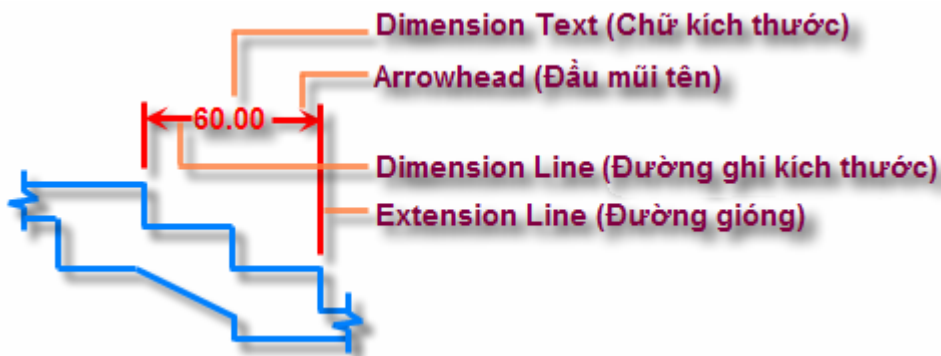
**GỢI Ý** Để xóa các đối tượng phi hình học (bao gồm cả đối tượng Linetype) không còn được sử dụng trong bản vẽ, có thể sử dụng phương thức PurgeAll có trong đối tượng Document, hoặc sử dụng lệnh Purge tại dòng lệnh của AutoCAD.

Đoạn mã sau thực hiện xóa kiểu đường “CENTER” khỏi bản vẽ AutoCAD.

```
Sub VD_LinetypeDelete ()
  Dim linetypeObj As AcadLineType
  Set linetypeObj = ThisDrawing.Linetypes("CENTER")
  linetypeObj.Delete
End Sub
```

### 5.7. Thao tác với đường kích thước – Dimension

Đường kích thước cho biết các số đo hình học của đối tượng như khoảng cách, góc giữa các đối tượng và tọa độ XY của một điểm. AutoCAD cung cấp 3 loại kích thước cơ bản: dạng đường (Kích thước dài, kích thước hình chiếu,...), dạng góc (kích thước góc,...) và dạng tia (kích thước bán kính, kích thước đường kính,...).



Hình V-15: Các thành phần cơ bản của đường kích thước.

Đường kích thước cũng là đối tượng hình học nên việc tạo các đường kích thước cũng tương tự như các đối tượng hình học khác. Trước tiên, xác định nơi chứa đối tượng (ModelSpace hay PaperSpace), sau đó sử dụng các phương thức tạo đường kích thước với dạng AddDimXXX (XXX là loại đường kích thước tương ứng).

Kiểu dáng của đường kích thước mới được tạo ra sẽ được lấy theo định dạng của kiểu đường kích thước (DimensionStyle) hiện hành trên bản vẽ. Sau khi được tạo ra, người dùng có thể thay đổi kiểu dáng của từng thành phần trong đường kích thước thông qua các thuộc tính tương ứng có trong đối tượng đường kích thước, hoặc có thể định dạng theo một kiểu đường kích thước đã có thông qua thuộc tính StyleName.

#### 5.7.1. Kiểu đường kích thước – DimensionStyle

Các kiểu đường kích thước trong một bản vẽ được quản lý trong tập đối tượng DimStyles của đối tượng Document. Mỗi kiểu đường kính thước, là một phần tử của tập đối tượng DimStyles, chứa các thiết lập chi tiết về kiểu dáng của đường kích thước.

**GỢI Ý** Trong AutoCAD, để thiết lập kiểu đường kích thước, người dùng có thể truy cập thông qua trình đơn **Format** ⇒ **Dimension Styles...**

### Tạo kiểu đường kích thước

Sử dụng phương thức `Add` có trong tập đối tượng `DimStyles` để tạo mới một kiểu đường kích thước. Cú pháp của phương thức này như sau:

```
SetRetVal = DimStyles.Add(Name)
```

Tham số	Mô tả
Name	Kiểu String. Tên của kiểu đường kích thước sẽ được tạo.
RetVal	Đối tượng kiểu DimStyle chứa kiểu đường kích thước vừa được tạo.

Đoạn mã sau sẽ tạo một kiểu đường kích thước mới có tên là "NewDimStyle"

```
Dim objDimStyle As AcadDimStyle
Set objDimStyle = ThisDrawing.DimStyles.Add("NewDimStyle")
```

### Định dạng kiểu đường kích thước

Kiểu đường kích thước mới được tạo sẽ thừa hưởng tất cả các thuộc tính của kiểu đường kích thước hiện hành trong bản vẽ. Tuy nhiên, việc hiệu chỉnh định dạng cho kiểu đường kích thước khá khó khăn vì đối tượng kiểu đường kích thước không có các thuộc tính hay phương thức phục vụ cho mục đích này. Để thay đổi định dạng cho kiểu đường kích thước, người dùng phải tiến hành gián tiếp thông qua việc thiết lập các biến hệ thống. Như vậy, việc tạo và hiệu chỉnh định dạng của một kiểu đường kích thước có thể được thực hiện theo trình tự sau:

1. Tạo đối tượng kiểu đường kích thước sử dụng phương thức `Add` có trong tập đối tượng `DimStyles`.
2. Thiết lập các biến hệ thống tương ứng với định dạng của từng thành phần trong kiểu đường kích thước cần thay đổi.
3. Sử dụng phương thức `CopyFrom` có trong đối tượng kiểu đường kích thước để cập nhật các định dạng đã thay đổi.

Ví dụ sau thực hiện tạo một kiểu đường kích thước mới có tên "NewDimStyle" và sau đó thực hiện hiệu chỉnh định dạng cho kiểu đường kích thước đó.

```
Public Sub NewDimStyle()
    Dim objDimStyle As AcadDimStyle
    ` Tạo kiểu đường kích thước mới
    Set objDimStyle = ThisDrawing.DimStyles.Add("NewDimStyle")

    `Thiết lập các biến hệ thống để định dạng kiểu đường kích thước
    ThisDrawing.SetVariable "DIMCLRD", acRed
    ThisDrawing.SetVariable "DIMCLRE", acBlue
    ThisDrawing.SetVariable "DIMCLRT", acWhite
    ThisDrawing.SetVariable "DIMLWD", acLnWtByLwDefault

    ` Cập nhật những thay đổi cho kiểu đường kích thước
    objDimStyle.CopyFrom ThisDrawing
End Sub
```

Có rất nhiều biến hệ thống dùng để định dạng kiểu đường kích thước và các biến hệ thống này đều được bắt đầu bằng tiền tố DIM. Người dùng có thể tham khảo các biến hệ thống này trong mục "System Variable" trong tài liệu "AutoCAD Command Reference" có sẵn trong bộ tài liệu

trợ giúp của AutoCAD. Dưới đây là danh sách các biến hệ thống định dạng kiểu đường kích thước hay được sử dụng:

Biến	Mô tả
DIMASZ	Kiểu Double. Xác định kích thước mũi tên của đường kích thước.
DIMBLK	Kiểu String. Xác định tên của loại mũi tên của đường kích thước. Một số giá trị thường dùng: "." → "_OBLIQUE" ↗ "_OPEN" →
DIMTSZ	Kiểu Double. Xác định kích thước của mũi tên trong trường hợp biến hệ thống DIMBLK được gán bằng "_OBLIQUE". Nếu gán bằng số khác không, kiểu mũi tên sẽ được tự động chuyển thành "_OBLIQUE"; nếu gán bằng 0, kiểu mũi tên sẽ là kiểu được xác định trong biến hệ thống DIMBLK.
DIMCLRDR	Kiểu Integer. Xác định màu của đường kích thước và mũi tên.
DIMCLRE	Kiểu Integer. Xác định màu của đường giống.
DIMCLRT	Kiểu Integer. Xác định màu của chữ kích thước.
DIMLWD	Kiểu Double. Xác định bề dày của đường kích thước
DIMDEC	Kiểu Integer. Xác định số chữ số sau dấu phẩy trong phần chữ kích thước.

### Sử dụng kiểu đường kích thước

Các đối tượng đường kích thước mới được tạo ra sẽ được lấy định dạng của kiểu đường kích thước hiện hành. Người dùng có thể thay đổi kiểu đường kích thước hiện hành bằng cách gán giá trị của thuộc tính `ActiveDimStyle` bằng một đối tượng kiểu đường kích thước. Chẳng hạn như khi muốn thay đổi kiểu đường kích thước hiện hành thành kiểu đường kích thước đã được tạo ở ví dụ trước, có thể dùng đoạn mã sau;

```
ThisDrawing.ActiveDimStyle = objDimStyle
```

Hoặc

```
ThisDrawing.ActiveDimStyle = ThisDrawing.DimStyles("NewDimStyle")
```

Ngoài ra, sau khi đường kích thước được tạo ra, người dùng có thể thay đổi kiểu đường kích thước cho nó bằng cách gán giá trị cho thuộc tính `StyleName` có trong đối tượng đường kích thước.

Ví dụ sau cho phép người dùng chọn một đường kích thước trên màn hình và thay đổi kiểu đường kích thước của đối tượng đó thành kiểu đường kích thước "NewDimStyle" đã được tạo ở ví dụ trước.

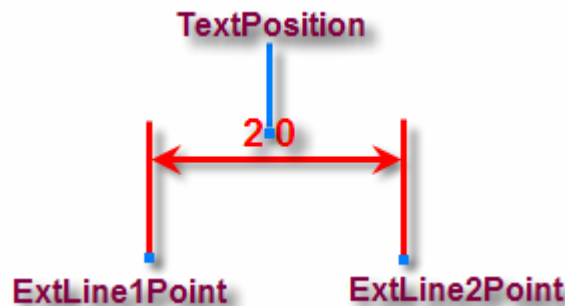
```
Sub VD_StyleName()
    Dim dimEnt As AcadEntity
    Dim P As Variant
    ' Chọn đối tượng đường kích thước trên màn hình
    ThisDrawing.Utility.GetEntity dimEnt, P, "Chọn đường kích thước: "
    ' Thay đổi kiểu đường kích thước cho đường kích thước được chọn
    dimEnt.StyleName = "NewDimStyle"
End Sub
```

### 5.7.2. Tạo đường kích thước

Để tạo mới đường kích thước, người dùng có thể sử dụng các phương thức `AddDimXXX`, với `xxx` là loại đường kích thước cần tạo. AutoCAD có nhiều loại đường kích thước khác nhau, dưới đây chỉ trình bày cách thức tạo các loại đường kích thước thường dùng.

### Đường kích thước dài - `DimAligned`

Sử dụng phương thức `AddDimAligned` để tạo mới đường kích thước dài. Với đường kích thước này, người dùng phải xác định 3 điểm: 2 điểm gốc và một điểm xác định vị trí chữ kích thước. Đường kích thước được tạo ra sẽ nằm song song với đoạn thẳng tạo bởi 2 điểm gốc.



Hình V-16: Đường kích thước dài.

Cú pháp của phương thức `AddDimAligned` như sau:

```
SetRetVal = object.AddDimAligned(ExtLine1Point, ExtLine2Point, TextPosition)
```

Tham số	Giải thích
<code>ExtLine1Point</code>	Kiểu Variant (mảng 3 phần tử kiểu Double). Tọa độ điểm gốc thứ nhất.
<code>ExtLine2Point</code>	Kiểu Variant (mảng 3 phần tử kiểu Double). Tọa độ điểm gốc thứ hai.
<code>TextPosition</code>	Kiểu Variant (mảng 3 phần tử kiểu Double). Tọa độ điểm ghi kích thước.
<code>RetVal</code>	Đối tượng kiểu <code>DimAligned</code> , tham chiếu đến đối tượng vừa mới được tạo.

Ví dụ sau sẽ tạo một đường kích thước dài với tọa độ hai điểm gốc là (5,5,0) và (10,8,0), còn tọa độ điểm ghi kích thước là (6.5,8,0)

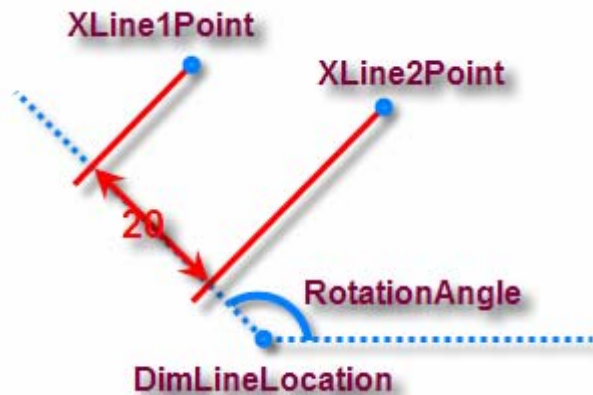
```
Sub VD_AddDimAligned()
    Dim dimObj As AcadDimAligned
    Dim P1(0 To 2) As Double
    Dim P2 (0 To 2) As Double
    Dim location(0 To 2) As Double

    ' Định nghĩa các điểm trên đường kích thước
    P1 (0) = 5#: P1 (1) = 5#: P1 (2) = 0#
    P2 (0) = 10#: P2 (1) = 8#: P2 (2) = 0#
    location(0) = 6.5: location(1) = 8#: location(2) = 0#

    ' Tạo đường kích thước dài trong không gian mô hình
    Set dimObj = ThisDrawing.ModelSpace.AddDimAligned(P1, P2, location)
    ZoomAll
End Sub
```

### Đường kích thước hình chiếu - `DimRotated`

Sử dụng phương thức `AddDimRotated` để tạo mới đường kích thước hình chiếu. Với đường kích thước này, người dùng cần phải xác định 2 điểm gốc, 1 điểm nằm trên đường kích thước và giá trị góc xoay. Kích thước được ghi là hình chiếu của đoạn thẳng nối 2 điểm gốc lên phương tạo với trục X một góc bằng góc xoay và đi qua điểm nằm trên đường kích thước. Hình dưới đây minh họa các thành phần cần thiết để tạo đường kích thước hình chiếu:



Hình V-17: Đường kích thước hình chiếu.

Cú pháp của phương thức `AddDimRotated` như sau:

```
SetRetVal = object.AddDimRotated(XLine1Point, XLine2Point, DimLineLocation, RotationAngle)
```

Tham số	Giải thích
XLine1Point	Kiểu Variant (mảng 3 phần tử kiểu Double). Tọa độ điểm gốc thứ nhất.
XLine2Point	Kiểu Variant (mảng 3 phần tử kiểu Double). Tọa độ điểm gốc thứ hai.
DimLineLocation	Kiểu Variant (mảng 3 phần tử kiểu Double). Tọa độ điểm nằm trên đường kích thước.
RotationAngle	Kiểu Double. Góc xoay so với phương ngang của phương chiếu, tính bằng Radian.
RetVal	Đối tượng kiểu DimRotated, tham chiếu đến đối tượng vừa mới được tạo.

Ví dụ sau tạo một đường kích thước hình chiếu với tọa độ hai điểm gốc là (0,5,0) và (5,5,0); tọa độ điểm nằm trên đường kích thước là (0,0,0); góc xoay phương chiếu bằng 120 độ.

```
Sub VD_AddDimRotated()
    Dim dimObj As AcadDimRotated
    Dim point1(0 To 2) As Double
    Dim point2(0 To 2) As Double
    Dim location(0 To 2) As Double
    Dim rotAngle As Double

    ' Xác định các giá trị cần thiết
    point1(0) = 0#: point1(1) = 5#: point1(2) = 0#
    point2(0) = 5#: point2(1) = 5#: point2(2) = 0#
    location(0) = 0#: location(1) = 0#: location(2) = 0#
    rotAngle = 120
    rotAngle = rotAngle * 3.141592 / 180#           ' Chuyển sang Radian

    ' Tạo đường kích thước hình chiếu trong không gian mô hình
    Set dimObj = ThisDrawing.ModelSpace.AddDimRotated _
        (point1, point2, location, rotAngle)
```

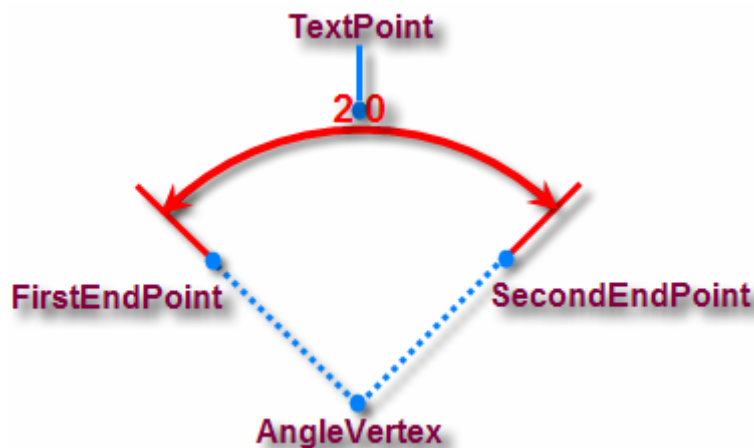


```
ZoomAll
End Sub
```

**GỢI Ý** Để tạo đường kích thước hình chiếu theo phương ngang (Horizontal) thì gán góc xoay phương chiếu bằng 0, còn đường kích thước hình chiếu theo phương đứng (Vertical) thì gán góc xoay phương chiếu bằng  $\text{Pi}/2$ .

### Đường kích thước góc – DimAngular

Sử dụng phương thức AddDimAngular để tạo mới đường kích thước góc. Với đường kích thước này, người dùng cần phải xác định tâm, hai điểm gốc và vị trí đặt chữ kích thước. Hình dưới đây minh họa các thành phần cần thiết để tạo đường kích thước góc:



Hình V-18: Đường kích thước góc

Cú pháp phương thức AddDimAngular như sau:

```
SetRetVal = object.AddDimAngular(AngleVertex, FirstEndPoint,
SecondEndPoint, TextPoint)
```

Tham số	Giải thích
AngleVertex	Kiểu Variant (mảng 3 phần tử kiểu Double). Toạ độ tâm của đường kích thước.
FirstEndPoint	Kiểu Variant (mảng 3 phần tử kiểu Double). Toạ độ điểm gốc thứ nhất.
SecondEndPoint	Kiểu Variant (mảng 3 phần tử kiểu Double). Toạ độ điểm gốc thứ hai.
TextPoint	Kiểu Variant (mảng 3 phần tử kiểu Double). Toạ độ điểm ghi kích thước.
RetVal	Đối tượng kiểu DimAngular, tham chiếu đến đối tượng vừa mới được tạo.

Ví dụ sau tạo đường kích thước đo góc trong không gian mô hình với toạ độ tâm là (0,5,0); toạ độ các điểm gốc là (1,7,0) và (1,3,0); toạ độ điểm ghi kích thước là (3,5,0):

```
Sub VD_AddDimAngular()
    Dim dimObj As AcadDimAngular
    Dim angVert(0 To 2) As Double
    Dim FirstPoint(0 To 2) As Double
    Dim SecondPoint(0 To 2) As Double
    Dim TextPoint(0 To 2) As Double

    ' Xác định các thông số cần thiết để tạo đường kích thước
    angVert(0) = 0#: angVert(1) = 5#: angVert(2) = 0#
```

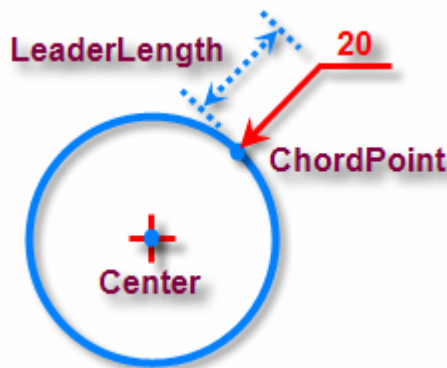
```

FirstPoint(0) = 1#: FirstPoint(1) = 7#: FirstPoint(2) = 0#
SecondPoint(0) = 1#: SecondPoint(1) = 3#: SecondPoint(2) = 0#
TextPoint(0) = 3#: TextPoint(1) = 5#: TextPoint(2) = 0#

' Tạo đường kích thước đo góc trong không gian mô hình
Set dimObj = ThisDrawing.ModelSpace.AddDimAngular(angVert,
FirstPoint, SecondPoint, TextPoint)
ZoomAll
End Sub
    
```

**Đường kích thước bán kính – DimRadial**

Sử dụng phương thức AddDimRadial để tạo đường kích thước bán kính. Để tạo được đường kích thước này, cần phải xác định được tọa độ tâm, tọa độ điểm đo (nằm trên cung tròn hoặc đường tròn) và chiều dài từ điểm đo đến chữ kích thước.



**Hình V-19: Đường kích thước bán kính.**

Cú pháp của phương thức AddDimRadial như sau:

```

SetRetVal = object.AddDimRadial(Center, ChordPoint, LeaderLength)
    
```

Tham số	Giải thích
Center	Kiểu Variant (mảng 3 phần tử kiểu Double). Tọa độ tâm của đường kích thước.
ChordPoint	Kiểu Variant (mảng 3 phần tử kiểu Double). Tọa độ điểm đo nằm trên đường tròn hoặc cung tròn.
LeaderLength	Kiểu Double. Khoảng cách từ chữ ghi kích thước đến điểm đo.
RetVal	Đối tượng kiểu DimRadial, tham chiếu đến đối tượng vừa mới được tạo.

Ví dụ sau tạo một đường kích thước bán kính trong không gian mô hình.

```

Sub VD_AddDimRadial()
Dim dimObj As AcadDimRadial
Dim center(0 To 2) As Double
Dim chordPoint(0 To 2) As Double
Dim leaderLen As Integer

' Xác định các thông số của đường kích thước
center(0) = 0#: center(1) = 0#: center(2) = 0#
chordPoint(0) = 5#: chordPoint(1) = 5#: chordPoint(2) = 0#
leaderLen = 2

' Tạo đường kích thước bán kính trong không gian mô hình
    
```

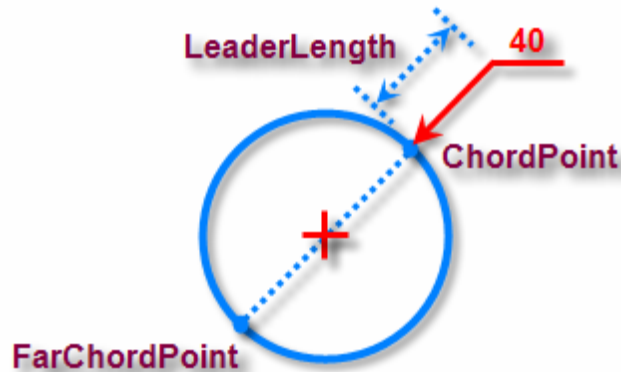
```

Set dimObj = ThisDrawing.ModelSpace.AddDimRadial _
              (center, chordPoint, leaderLen)

ZoomAll
End Sub
    
```

### Đường kích thước đường kính – DimDiametric

Sử dụng phương thức AddDimDiametric để tạo mới đường kích thước đường kính. Để tạo được đường kích thước này, cần phải xác định 2 điểm đo nằm trên đường tròn và khoảng cách từ điểm đo thứ nhất đến chữ ghi kích thước.



Hình V-20: Đường kích thước đường kính.

Cú pháp của phương thức AddDimDiametric như sau:

```

Set RetVal = object.AddDimDiametric(ChordPoint, FarChordPoint,
LeaderLength)
    
```

Tham số	Giải thích
ChordPoint	Kiểu Variant (mảng 3 phần tử kiểu Double). Toạ độ điểm đo thứ nhất nằm trên đường tròn hoặc cung tròn.
FarChordPoint	Kiểu Variant (mảng 3 phần tử kiểu Double). Toạ độ điểm đo thứ hai nằm trên đường tròn hoặc cung tròn.
LeaderLength	Kiểu Double. Khoảng cách từ chữ ghi kích thước đến điểm đo thứ nhất.
RetVal	Đối tượng kiểu DimDiametric, tham chiếu đến đối tượng vừa mới được tạo.

Ví dụ sau minh hoạ cách thức sử dụng phương thức AddDimDiametric.

```

Sub VD_AddDimDiametric()
    Dim dimObj As AcadDimDiametric
    Dim chordPoint(0 To 2) As Double
    Dim farChordPoint(0 To 2) As Double
    Dim leaderLength As Double

    ' Xác định các thông số của đường kích thước
    chordPoint(0) = 5#: chordPoint(1) = 3#: chordPoint(2) = 0#
    farChordPoint(0) = 5#: farChordPoint(1) = 5#: farChordPoint(2) = 0#
    leaderLength = 2#

    ' Tạo đường kích thước đường kính trong không gian mô hình
    Set dimObj = ThisDrawing.ModelSpace.AddDimDiametric _
    
```

```
(chordPoint, farChordPoint, leaderLength)
ZoomAll
End Sub
```

**5.7.3. Định dạng đường kích thước**

Ngoài cách định dạng đường kích thước bằng kiểu đường kích thước thông qua thuộc tính StyleName như đã trình bày ở mục “Sử dụng kiểu đường kích thước” trang 261, người dùng còn có thể thay đổi các định dạng này thông qua các thuộc tính tương ứng của đối tượng đường kích thước. Dưới đây là danh sách các thuộc tính để định dạng cho đường kích thước thường dùng:

Thuộc tính	Mô tả
AngleFormat	Quy định định dạng đơn vị của kích thước dạng góc.
Arrowhead1Type, Arrowhead2Type	Quy định dạng đầu mũi tên của đường kích thước.
ArrowheadSize	Quy định cỡ đầu mũi tên của đường kích thước.
CenterMarkSize	Quy định cỡ của dấu tâm cho các kích thước dạng tia (đường kích thước góc, bán kính, đường kính,...)
CenterType	Quy định dạng của dấu tâm cho kích thước dạng tia.
DecimalSeparator	Quy định ký tự dùng làm dấu cách thập phân.
DimensionLineColor	Quy định màu cho đường ghi kích thước.
DimensionLineWeight	Quy định độ dày của đường ghi kích thước.
ExtensionLineColor	Quy định màu của các đường giống.
ExtensionLineExtend	Quy định khoảng cách từ đường giống đến đường ghi kích thước.
ExtensionLineOffset	Quy định khoảng cách từ đường giống đến điểm gốc của đường giống.
ExtensionLineWeight	Quy định độ dày của đường giống.
LinearScaleFactor	Quy định hệ số tỷ lệ toàn cục cho các số đo kích thước dạng đường.
PrimaryUnitsPrecision	Quy định số chữ số thập phân hiển thị trong đơn vị chính của kích thước.
TextColor	Quy định màu của chữ kích thước.
TextHeight	Quy định độ cao của chữ kích thước.
TextRotation	Quy định góc nghiêng của chữ kích thước.

**5.8. Thao tác với dữ liệu mở rộng – XData**

Ngoài các thuộc tính có trong mỗi đối tượng, AutoCAD còn cho phép tạo thêm các thuộc tính mới để lưu trữ các thông tin do người lập trình tự định nghĩa. Những thông tin này sẽ được AutoCAD lưu cùng với đối tượng trong bản vẽ.

Mỗi đối tượng có thể chứa nhiều dữ liệu mở rộng khác nhau. Thông thường, các dữ liệu mở rộng được bắt đầu bằng tên của ứng dụng, tiếp đến là các dữ liệu khác. Để xác định các dữ liệu mở rộng, phải sử dụng hai mảng có chiều dài bằng nhau, một mảng kiểu Short xác định kiểu dữ liệu của dữ liệu mở rộng, mảng thứ hai là mảng kiểu Variant chứa các dữ liệu tương ứng.

**5.8.1. Gán dữ liệu mở rộng**

Sử dụng phương thức SetXData để gán các dữ liệu mở rộng cho đối tượng. Cú pháp của phương thức này như sau:

```
object.SetXData XDataType, Xdata
```

Tham số	Giải thích
Object	Là đối tượng sẽ được gán dữ liệu mở rộng.
XDataType	Mảng kiểu Short, xác định kiểu dữ liệu của dữ liệu mở rộng
Xdata	Mảng kiểu Variant, xác định giá trị của dữ liệu mở rộng

Khi gán giá trị cho mảng XDataType và Xdata, cần phải lưu ý:

- ✦ Cả hai mảng đều là mảng một chiều và phải có kích thước bằng nhau;
- ✦ Giá trị phần tử của mảng Xdata phải có kiểu dữ liệu tương ứng với kiểu dữ liệu được xác định trong mảng XDataType.

Bảng dưới đây là danh sách các giá trị thường dùng trong mảng XDataType và ý nghĩa tương ứng:

Giá trị	Ý nghĩa
1001	Chuỗi chứa tên của ứng dụng. Tên của ứng dụng là do người lập trình tự thiết lập.
1000	Giá trị kiểu String.
1003	Tên của Layer.
1010	Toạ độ 3D của một điểm
1040	Giá trị của Double.
1071	Giá trị kiểu Interger.

Ví dụ sau sẽ tạo một đường thẳng trong không gian mô hình, sau đó tiến hành gán các dữ liệu mở rộng cho đường thẳng đó.

```
Sub VD_SetXdata()
    ' Tạo đường thẳng
    Dim lineObj As AcadLine
    Dim startPt(0 To 2) As Double, endPt(0 To 2) As Double
    startPt(0) = 1#: startPt(1) = 1#: startPt(2) = 0#
    endPt(0) = 5#: endPt(1) = 5#: endPt(2) = 0#
    Set lineObj = ThisDrawing.ModelSpace.AddLine(startPt, endPt)

    ZoomAll

    ' Khởi tạo các giá trị cho dữ liệu mở rộng.
    ' Chú ý là dữ liệu đầu tiên phải là tên của ứng dụng
    ' và mã tương ứng là 1001
    Dim DataType(0 To 5) As Integer      'Mảng chứa kiểu dữ liệu
    Dim Data(0 To 5) As Variant          'Mảng chứa dữ liệu
    Dim reals3(0 To 2) As Double

    DataType(0) = 1001: Data(0) = "Test_Application"
    DataType(1) = 1000: Data(1) = "This is a test for xdata"
    DataType(2) = 1003: Data(2) = "0"                                     ' Tên lớp
    DataType(3) = 1040: Data(3) = 1.23479137438413E+40                 ' Kiểu Double
    DataType(4) = 1071: Data(4) = 32767                                ' Kiểu Integer
    reals3(0) = -2.95: reals3(1) = 100: reals3(2) = -20
    DataType(5) = 1010: Data(5) = reals3                               ' Toạ độ điểm

    ' Gán dữ liệu mở rộng vào đường thẳng
    lineObj.SetXData DataType, Data

End Sub
```

## 5.8.2. Đọc dữ liệu mở rộng

Sử dụng phương thức GetXData để đọc dữ liệu mở rộng có trong một đối tượng. Cú pháp của phương thức này như sau:

```
object.GetXData AppName, XDataType, XdataValue
```

Tham số	Giải thích
Object	Là đối tượng có chứa dữ liệu mở rộng.
AppName	Là chuỗi chứa tên của ứng dụng (như đã được gán khi sử dụng phương thức SetXData). Nếu tham số AppName là một chuỗi rỗng, phương thức này sẽ trả về tất cả các dữ liệu mở rộng có trong đối tượng. Nếu có truyền giá trị vào tham số AppName, phương thức này chỉ trả về dữ liệu mở rộng được tạo bởi ứng dụng có tên như đã xác định trong tham số AppName.
XDataType	Mảng kiểu Short, xác định kiểu dữ liệu của dữ liệu mở rộng được trả về.
Xdata	Mảng kiểu Variant, xác định giá trị của dữ liệu mở rộng được trả về.

Ví dụ sau cho phép người dùng chọn một đối tượng hình học trên bản vẽ, sau đó sẽ hiển thị tất cả các dữ liệu mở rộng của đối tượng được chọn (nếu có)

```
Sub VD_GetXData()
    Dim sset As AcadSelectionSet

    On Error Resume Next
    Set sset = ThisDrawing.SelectionSets("MySSet")
    sset.Delete
    Set sset = ThisDrawing.SelectionSets.Add("MySSet")

    ThisDrawing.Utility.Prompt vbCrLf & "Chon doi tuong can xem Xdata: "
    sset.SelectOnScreen

    Dim ent As AcadEntity
    Dim XDataType As Variant

    Dim XData As Variant

    Dim i As Integer
    For Each ent In sset
        ent.GetXData "", XDataType, XData
        If Not IsEmpty(XDataType) Then
            ThisDrawing.Utility.Prompt (vbCrLf & ent.ObjectName)
            For i = LBound(XDataType) To UBound(XDataType)
                ThisDrawing.Utility.Prompt vbCrLf & XDataType(i)
                ThisDrawing.Utility.Prompt " : " & XData(i)
            Next i
        Else
            ThisDrawing.Utility.Prompt vbCrLf & "Doi tuong khong chua XData"
        End If
    Next ent
End Sub
```

## 6. Giao diện người dùng

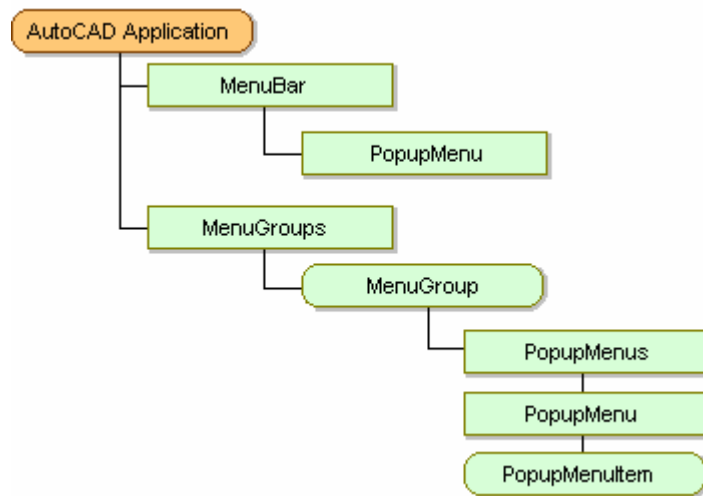
Quá trình nhập dữ liệu và tương tác với người dùng được hỗ trợ theo nhiều cách khác nhau trong AutoCAD:

- ✦ Thông qua việc nhập liệu tại dòng lệnh: người dùng có thể nhập vào một chuỗi, một số, hoặc thực hiện một chọn lựa nào đó... Các thao tác nhập liệu tại dòng lệnh tham khảo thêm mục “*Nhập dữ liệu người dùng từ dòng lệnh của AutoCAD*” trang 207
- ✦ Thông qua việc tương tác trực tiếp trên bản vẽ: người dùng thường sẽ thực hiện lựa chọn đối tượng, xác định toạ độ điểm,... Các thao tác tương tác trực tiếp trên bản vẽ tham khảo thêm mục “*Làm việc với đối tượng SelectionSet*” trang 227 hoặc mục “*Nhập dữ liệu người dùng từ dòng lệnh của AutoCAD*” trang 207.
- ✦ Thông qua hộp thoại tùy biến – Userform: quá trình nhập dữ liệu được thực hiện hầu hết trên Userform, các chức năng của chương trình sẽ được trình bày trên UserForm thông qua các nút bấm,... Các thông tin về cách thức lập trình trên UserForm, tham khảo thêm mục “*Làm việc với UserForm và các thành phần điều khiển*” trang 60 và mục “*Hộp thoại tùy biến – UserForm*” trang 169.
- ✦ Thông qua hệ thống thanh trình đơn và thanh công cụ: người dùng có thể thực hiện một chức năng nào đó của chương trình thông qua việc chọn một mục trình đơn tương ứng hoặc chọn một nút lệnh trên thanh công cụ. Phần này sẽ tập trung giới thiệu cách thức tạo trình đơn trong AutoCAD.

## 6.1. Thao tác với thanh trình đơn

### 6.1.1. Cấu trúc của hệ thống thanh trình đơn

Hệ thống trình đơn trong AutoCAD được tổ chức theo cấu trúc phân cấp. Mô hình đối tượng của hệ thống thanh trình đơn trong AutoCAD như sau:

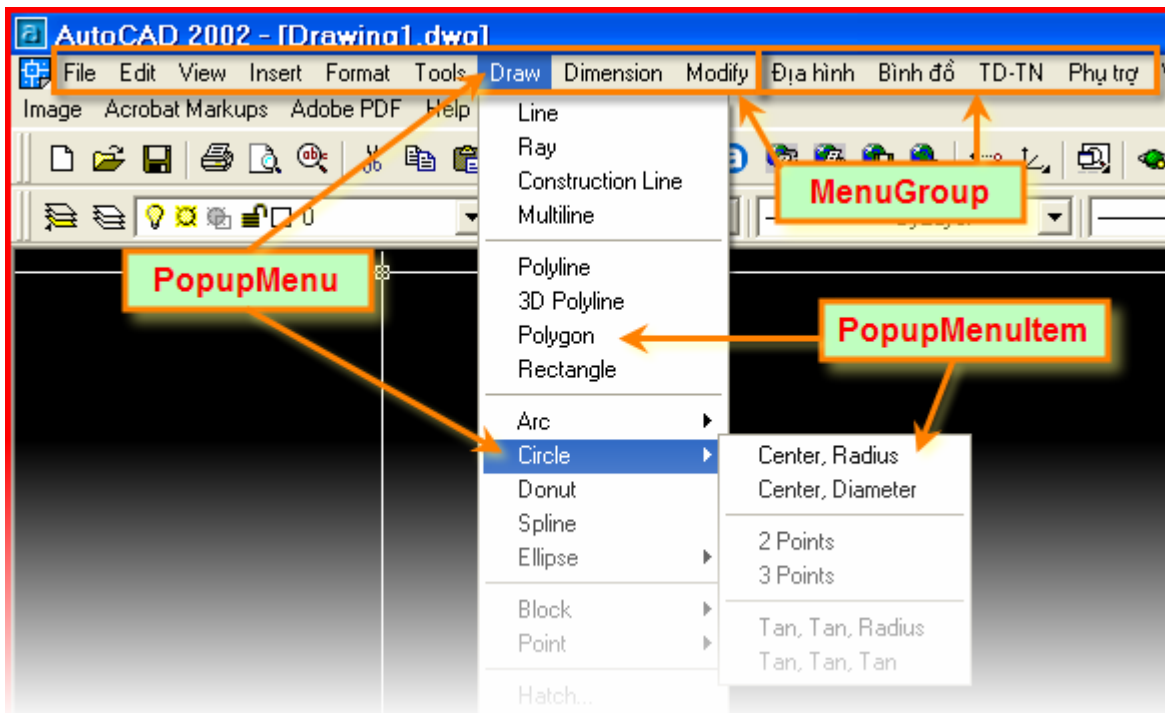


**Hình V-21: Mô hình đối tượng của hệ thống thanh trình đơn trong AutoCAD**

- ✦ MenuBar là thanh trình đơn nằm ngay phía dưới thanh tiêu đề của cửa sổ chương trình AutoCAD. Trong MenuBar có chứa các PopupMenu, là một trình đơn xổ xuống khi người dùng kích chuột vào, chẳng hạn như File, Edit, View,...
- ✦ MenuGroups là tập đối tượng chứa các MenuGroup, là nhóm các trình đơn trong AutoCAD. Thông thường, mỗi chương trình đều tạo cho mình một MenuGroup riêng. Trong mỗi MenuGroup sẽ có chứa các PopupMenu. Tập hợp tất cả các PopupMenu trong tất cả các MenuGroup sẽ hình thành nên MenuBar trên màn hình của chương trình AutoCAD. Tuy nhiên, không phải tất cả các PopupMenu trong MenuGroup đều được hiển thị trên MenuBar, chỉ khi nào người lập trình hoặc người dùng thêm vào MenuBar, MenuPopup đó mới được hiển thị trên MenuBar.
- ✦ PopupMenuItem là thành phần cuối cùng trong cấu trúc phân cấp đối tượng của thanh trình đơn. Mỗi PopupMenuItem chính là một lệnh trong hệ thống thanh trình đơn. Khi

người dùng kích chuột vào PopupMenuItem thì AutoCAD sẽ thực hiện một chức năng nào đó của chương trình.

Các thành phần của hệ thống trình đơn được minh họa ở hình sau:



Hình V-22: Các đối tượng trong hệ thống trình đơn của AutoCAD

### 6.1.2. Tạo trình đơn

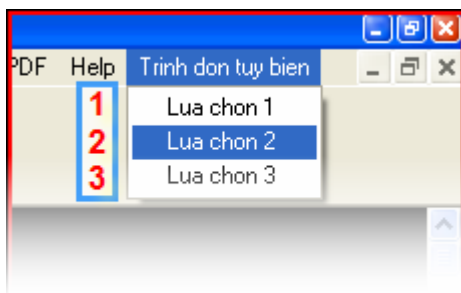
Người lập trình có thể dễ dàng thêm và hiệu chỉnh hệ thống trình đơn trong Excel thông qua các đoạn mã lệnh bằng VBA theo các bước sau:

1. Phác thảo trình đơn tùy biến cần tạo và các chức năng tương ứng.
2. Viết mã lệnh cho từng PopupMenuItem. Mỗi đoạn mã lệnh này được chứa trong một Macro.
3. Tham chiếu đến MenuGroup, nơi cần tạo trình đơn tùy biến.
4. Tạo PopupMenu và PopupMenuItem.
5. Gán các đoạn mã lệnh tương ứng đã tạo ở bước 2 cho từng PopupMenuItem.

Dưới đây sẽ lần lượt trình bày các bước để tạo trình đơn tùy biến trong AutoCAD

### Ví dụ tạo trình đơn

1. Phác thảo cấu trúc trình đơn như sau:



2. Viết mã lệnh cho từng PopupMenuItem



```
'MÃ LỆNH CHO "LUA CHON 1"
Sub Macro1()
  MsgBox "Ban da chon Lua chon 1"
End Sub
'MÃ LỆNH CHO "LUA CHON 2"
Sub Macro2()
  MsgBox "Ban da chon Lua chon 2"
End Sub
'MÃ LỆNH CHO "LUA CHON 3"
Sub Macro3()
  MsgBox "Ban da chon Lua chon 3"
End Sub
```

3. Viết mã lệnh để: tham chiếu đến MenuGroup, tạo trình đơn tùy biến và gán mã lệnh tương ứng

```
Sub VD_TaoMenu()
  ' Định nghĩa biến và tham chiếu đến MenuGroup đầu tiên
  Dim currMenuGroup As AcadMenuGroup
  Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)

  ' Tạo trình đơn (Tạo PopupMenu)
  Dim newMenu As AcadPopupMenu
  Set newMenu = currMenuGroup.Menus.Add("Trinh don tuy bien")

  ' Khai báo biến cho PopupMenuItem
  Dim newItem As AcadPopupMenu
  Dim openMacro As String

  ' Tạo PopupMenuItem và gán mã lệnh tương ứng
  openMacro = "-vbarun Macro1 "
  Set newItem = newMenu.AddMenuItem(newMenu.Count + 1, _
    "Lua chon 1", openMacro)

  openMacro = "-vbarun Macro2 "
  Set newItem = newMenu.AddMenuItem(newMenu.Count + 1, _
    "Lua chon 2", openMacro)

  openMacro = "-vbarun Macro3 "
  Set newItem = newMenu.AddMenuItem(newMenu.Count + 1, _
    "Lua chon 3", openMacro)

  ' Hiện thị trình đơn vừa tạo trên thanh trình đơn (MenuBar)
  currMenuGroup.Menus.InsertMenuInMenuBar "Trinh don tuy bien", ""
End Sub
```

Các đoạn mã được tạo trong bước 2 và bước 3 phải được lưu trong cùng một mô-đun chuẩn của dự án VBA.

Tuy nhiên, khi đã tạo được trình đơn, nếu thực thi Macro ở bước 3 một lần nữa thì chương trình sẽ báo lỗi. Nguyên nhân là do "Trinh don tuy bien" đã được tạo ở lần thực thi Macro trước. Chính vì vậy, để khắc phục lỗi này, cần phải thay đoạn mã lệnh ở bước 3 bằng đoạn mã lệnh sau:

```
Sub VD_TaoMenu2()
  ' Định nghĩa biến và tham chiếu đến MenuGroup đầu tiên
  Dim currMenuGroup As AcadMenuGroup
```

```

Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)

' Tạo trình đơn (Tạo PopupMenu)
' và xử lý tình huống khi trình đơn đã được tạo
Dim newMenu As AcadPopupMenu
On Error Resume Next
Set newMenu = currMenuGroup.Menus.Add("Trình đơn tùy biến")
If Err <> 0 Then
    Set newMenu = currMenuGroup.Menus("Trình đơn tùy biến")
    Dim menuEnt As AcadPopupMenuEnt
    For Each menuEnt In newMenu
        menuEnt.Delete
    Next
End If

' Khai báo biến cho PopupMenuItem
Dim newItem As AcadPopupMenuEnt
Dim openMacro As String

' Tạo PopupMenuItem và gán mã lệnh tương ứng
openMacro = "-vbarun Macro1 "
Set newItem = newMenu.AddMenuItem(newMenu.Count + 1, _
    "Lua chọn 1", openMacro)

openMacro = "-vbarun Macro2 "
Set newItem = newMenu.AddMenuItem(newMenu.Count + 1, _
    "Lua chọn 2", openMacro)

openMacro = "-vbarun Macro3 "

Set newItem = newMenu.AddMenuItem(newMenu.Count + 1, _
    "Lua chọn 3", openMacro)

' Hiển thị trình đơn vừa tạo trên thanh trình đơn (MenuBar)
currMenuGroup.Menus.InsertMenuInMenuBar "Trình đơn tùy biến", ""
End Sub

```

### **6.1.3. Xóa thanh trình đơn**

Các trình đơn tùy biến trong AutoCAD sẽ được tự động xóa đi khi khởi động lại AutoCAD. Tuy nhiên, khi cần thiết, người lập trình có thể gỡ bỏ các trình đơn tùy biến ra khỏi AutoCAD bằng mã lệnh VBA.

Cần lưu ý là không thể xóa hẳn trình đơn tùy biến ra khỏi AutoCAD bằng mã lệnh VBA mà chỉ có thể gỡ bỏ trình đơn tùy biến ra khỏi thanh trình đơn của AutoCAD. Chỉ cần khởi động lại AutoCAD, trình đơn tùy biến sẽ được tự động xóa khỏi AutoCAD.

Để gỡ bỏ trình đơn tùy biến có tên là "Trình đơn tùy biến" đã được tạo ở ví dụ trước, sử dụng đoạn mã sau:

```

Sub VD_XoaMenu()
    Dim pMenu As AcadPopupMenu
    On Error Resume Next
    Set pMenu = Application.MenuBar("Trình đơn tùy biến")

    ' Kiểm tra xem nếu trình đơn tùy biến đã có thì sẽ
    ' gỡ bỏ khỏi thanh trình đơn của AutoCAD
    If Not (pMenu Is Nothing) Then
        pMenu.RemoveFromMenuBar
    End If
End Sub

```

```
End If  
End Sub
```

## PHẦN III: TÀI LIỆU THAM KHẢO

---

1. **John Walkenbach** – Excel 2002 Power Programming with VBA – M&T Books – 2001 .
2. **Steve Saunders, Jeff Webb** – Programming Excel with VBA and .NET – O'Reilly – 2006.
3. **Richard Shepherd** – Excel VBA Macro Programming – McGraw-Hill – 2004.
4. **Autodesk®** (Người dịch: **Lê Quỳnh Mai, Trương Thanh Hoàng, Hoàng Thuỳ Linh**) – Phát triển AutoCAD bằng ActiveX & VBA – 2006.
5. **Joe Sutphin** – AutoCAD 2006 VBA: A Programmer's Reference – Apress® – 2005.