MÔN TIN HỌC

Đối tượng : SV đại học chính quy toàn trường

Nội dung chính gồm 12 chương :

1. Phương pháp giải quyết bài toán bằng máy tính số.
2. Thể hiện dữ liệu trong máy tính số.
3. Tổng quát về lập trình bằng VB.
4. Qui trình thiết kế trực quan giao diện.
5. Các kiểu dữ liệu của VB.
6. Các lệnh định nghĩa & khai báo.
7. Biểu thức VB.
8. Các lệnh thực thi VB.
9. Định nghĩa thủ tục & sử dụng.
10. Tương tác giữa người dùng & chương trình.
11. Quản lý hệ thống file.
12. Linh kiện phần mềm & truy xuất database.

Tài liệu tham khảo :

- Tập slide bài giảng & thực hành của môn học này.
- 3 CD MSDN trong Microsoft Visual Studio.

MÔN TIN HỌC

Chương 1

PHƯƠNG PHÁP GIẢI QUYẾT BÀI TOÁN BẰNG MÁY TÍNH SỐ

1.1 Các khái niệm cơ bản về máy tính số

- Con người thông minh hơn các động vật khác nhiều. Trong cuộc sống, họ đã chế tạo ngày càng nhiều công cụ, thiết bị để hỗ trợ mình trong hoạt động. Các công cụ, thiết bị do con người chế tạo ngày càng tinh vi, phức tạp và thực hiện nhiều công việc hơn trước đây. Mỗi công cụ, thiết bị thường chỉ thực hiện được 1 vài công việc cụ thể nào đó. Thí dụ, cây chổi để quét, radio để bắt và nghe đài audio...
- Máy tính số (digital computer) cũng là 1 thiết bị, nhưng thay vì chỉ thực hiện 1 số chức năng cụ thể, sát với nhu cầu đời thường của con người, nó có thể thực hiện 1 số hữu hạn các chức năng cơ bản (tập lệnh), mỗi lệnh rất sơ khai chưa giải quyết trực tiếp được nhu cầu đời thường nào của con người. Cơ chế thực hiện các lệnh là tự động, bắt đầu từ lệnh được chỉ định nào đó rồi tuần tự từng lệnh kế tiếp cho đến lệnh cuối cùng. Danh sách các lệnh được thực hiện này được gọi là chương trình.

Các khái niệm cơ bản về máy tính số

- Các lệnh mà máy hiểu và thực hiện được được gọi là lệnh máy. Ta dùng ngôn ngữ để miêu tả các lệnh. Ngôn ngữ lập trình cấu thành từ 2 yếu tố chính yếu : cú pháp và ngữ nghĩa. Cú pháp qui định trật tự kết hợp các phần tử để cấu thành 1 lệnh (câu), còn ngữ nghĩa cho biết ý nghĩa của lệnh đó.
- Bất kỳ công việc (bài toán) ngoài đời nào cũng có thể được chia thành trình tự nhiều công việc nhỏ hơn. Trình tự các công việc nhỏ này được gọi là giải thuật giải quyết công việc ngoài đời. Mỗi công việc nhỏ hơn cũng có thể được chia nhỏ hơn nữa nếu nó còn phức tạp,... $\Rightarrow$ công việc ngoài đời có thể được miêu tả bằng 1 trình tự các lệnh máy (chương trình ngôn ngữ máy).

Các khái niệm cơ bản về máy tính số

- Vấn đề mấu chốt của việc dùng máy tính giải quyết công việc ngoài đời là lập trình (được hiểu nôm na là qui trình xác định trình tự đúng các lệnh máy để thực hiện công việc). Cho đến nay, lập trình là công việc của con người (với sự trợ giúp ngày càng nhiều của máy tính).
- Với công nghệ phần cứng hiện nay, ta chỉ có thể chế tạo các máy tính mà tập lệnh máy rất sơ khai, mỗi lệnh máy chỉ có thể thực hiện 1 công việc rất nhỏ và đơn giản ⇒ công việc ngoài đời thường tương đương với trình tự rất lớn (hàng triệu) các lệnh máy ⇒ Lập trình bằng ngôn ngữ máy rất phức tạp, tốn nhiều thời gian, công sức, kết quả rất khó bảo trì, phát triển.
- Ta muốn có máy luận lý với tập lệnh (được đặc tả bởi ngôn ngữ lập trình) cao cấp và gần gủi hơn với con người. Ta thường hiện thực máy này bằng 1 máy vật lý + 1 chương trình dịch. Có 2 loại chương trình dịch : trình biên dịch (compiler) và trình thông dịch (interpreter).

Trình biên dịch (Compiler)

- Chương trình biên dịch nhận một chương trình nguồn (thường được viết bằng ngôn ngữ cấp cao) và tạo ra một chương trình đối tượng tương ứng về chức năng nhưng thường được viết bằng ngôn ngữ cấp thấp (thường là ngôn ngữ máy).
- Nếu có lỗi xảy ra trong lúc dịch, trình biên dịch sẽ báo lỗi, cố gắng tìm vị trí đúng kế tiếp rồi tiếp tục dịch… Nhờ vậy, mỗi lần dịch 1 chương trình, ta sẽ xác định được nhiều lỗi nhất có thể có.
- Sau mỗi lần dịch, nếu không có lỗi, trình biên dịch sẽ tạo ra file chứa chương trình đối tượng (thí dụ file chương trình khả thi *.exe trên Windows).
- Để chạy chương trình, người dùng chỉ cần kích hoạt file khả thi (người dùng không biết và không cần quan tâm đến file chương trình nguồn).

Trình thông dịch (Interpreter)

- Chương trình thông dịch không tạo ra và lưu giữ chương trình đối tượng.
- Mỗi lần thông dịch 1 chương trình nguồn là 1 lần cố gắng chạy chương trình này theo cách thức sau :
  - dịch và chuyển sang mã thực thi từng lệnh một rồi nhờ máy chạy đoạn lệnh tương ứng.
  - Nếu có lỗi thì báo lỗi, nếu không có lỗi thì thông dịch lệnh kế tiếp... cho đến khi hết chương trình.
  - Như vậy, mỗi lần thông dịch chương trình, trình thông dịch chỉ thông dịch các lệnh trong luồng thi hành cần thiết chứ không thông dịch hết mọi lệnh của chương trình nguồn. Do đó, sau khi thông dịch thành công 1 chương trình, ta không thể kết luận rằng chương trình này không có lỗi.

So sánh trình biên dịch & trình thông dịch

- Mọi hoạt động xử lý trên mọi mã nguồn của chương trình (kiểm tra lỗi, dịch ra các lệnh đối tượng tương đương,...) đều được chương trình biên dịch thực hiện để tạo được chương trình đối tượng. Do đó sau khi dịch các file mã nguồn của chương trình, nếu không có lỗi, ta có thể kết luận chương trình không thể có lỗi thời điểm dịch (từ vựng, cú pháp). Quá trình biên dịch và quá trình thực thi chương trình là tách rời nhau : biên dịch 1 lần và chạy nhiều lần cho đến khi cần cập nhật version mới của chương trình.
- Chương trình thông dịch sẽ thông dịch từng lệnh theo luồng thi hành của chương trình bắt đầu từ điểm nhập của chương trình, thông dịch 1 lệnh gồm 2 hoạt động : biên dịch lệnh đó và thực thi các lệnh kết quả. Nếu 1 đoạn lệnh cần được thực thi lặp lại thì trình thông dịch sẽ phải thông dịch lại tất cả đoạn lệnh đó. Điều này sẽ làm cho việc chạy chương trình trong chế độ thông dịch không hiệu quả.
- Việc chạy chương trình bằng cơ chế thông dịch đòi hỏi chương trình thông dịch và chương trình ứng dụng cần chạy phải tồn tại đồng thời trong bộ nhớ máy tính, do đó có nguy cơ chạy không được các chương trình lớn nếu tài nguyên của máy không đủ cho cả 2 chương trình thông dịch và chương trình ứng dụng.

Các khái niệm cơ bản về máy tính số

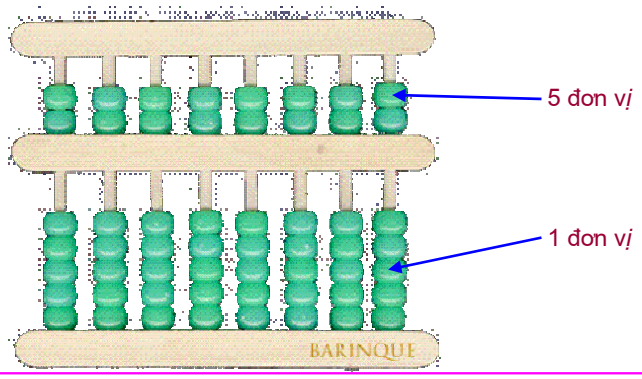
- Gọi ngôn ngữ máy vật lý là N_0. Trình biên dịch ngôn ngữ N_1 sang ngôn ngữ N_0 sẽ nhận đầu vào là chương trình được viết bằng ngôn ngữ N_1, phân tích từng lệnh N_1 rồi chuyển thành danh sách các lệnh ngôn ngữ N_0 có chức năng tương đương. Để viết chương trình dịch từ ngôn ngữ N_1 sang N_0 dễ dàng, độ phức tạp của từng lệnh ngôn ngữ N_1 không quá cao so với từng lệnh ngôn ngữ N_0.
- Sau khi có máy luận lý hiểu được ngôn ngữ luận lý N_1, ta có thể định nghĩa và hiện thực máy luận lý N_2 theo cách trên và tiếp tục đến khi ta có 1 máy luận lý hiểu được ngôn ngữ N_m rất gần g**ũi** với con người, dễ dàng miêu tả giải thuật của bài toán cần giải quyết...
- Nhưng qui trình trên chưa có điểm dừng, với yêu cầu ngày càng cao và kiến thức ngày càng nhiều, người ta tiếp tục định nghĩa những ngôn ngữ mới với tập lệnh ngày càng gần g**ũi** hơn với con người để miêu tả giải thuật càng dễ dàng, gọn nhẹ và trong sáng hơn.

Các cấp độ ngôn ngữ lập trình

- Ngôn ngữ máy vật lý là loại ngôn ngữ thấp nhất mà người lập trình bình thường có thể dùng được. Các lệnh và tham số của lệnh được miêu tả bởi các số binary (hay hexadecimal - sẽ được miêu tả chi tiết trong chương 2). Đây là loại ngôn ngữ mà máy vật lý có thể hiểu trực tiếp, nhưng con người thì gặp nhiều khó khăn trong việc viết và bảo trì chương trình ở cấp này.
- Ngôn ngữ assembly rất gần với ngôn ngữ máy, những lệnh cơ bản nhất của ngôn ngữ assembly tương ứng với lệnh máy nhưng được biểu diễn dưới dạng gợi nhớ. Ngoài ra, người ta tăng cường thêm khái niệm "lệnh macro" để nâng sức mạnh miêu tả giải thuật.
- Ngôn ngữ cấp cao theo trường phái lập trình cấu trúc như Pascal, C,... Tập lệnh của ngôn ngữ này khá mạnh và gần với tư duy của người bình thường.
- Ngôn ngữ hướng đối tượng như C++, Visual Basic, Java, C#,... cải tiến phương pháp cấu trúc chương trình sao cho trong sáng, ổn định, dễ phát triển và thay thế linh kiện.

1.2 Lịch sử phát triển máy tính số

- Máy tính xuất hiện từ rất lâu theo nhu cầu buôn bán và trao đổi tiền tệ.
- Bàn tính tay abacus là dạng sơ khai của máy tính.



Các thế hệ máy tính số

Blaise Pascal (Pháp-1642)

Charles Babbage (Anh-1830)

ENIAC (1946)
18.000 bóng đèn
1500 rờ le
30 tấn
140 KW

Intel 8080 (1974)
được xem như CPU đầu tiên được tích hợp trên 1 chip

IBM 360 (1965)

Von Neumann (1945)

PDP-1 (1961)

80x86 (1978)

Cơ

Đèn điện tử

Transistors

IC

?

(1642 - 1945)

(1945 - 1955)

(1955 - 1965)

(1965 - 1980)

(1980 - ????)

Herman Hollerith lập IBM (International Business Machine) ở Mỹ - 1890

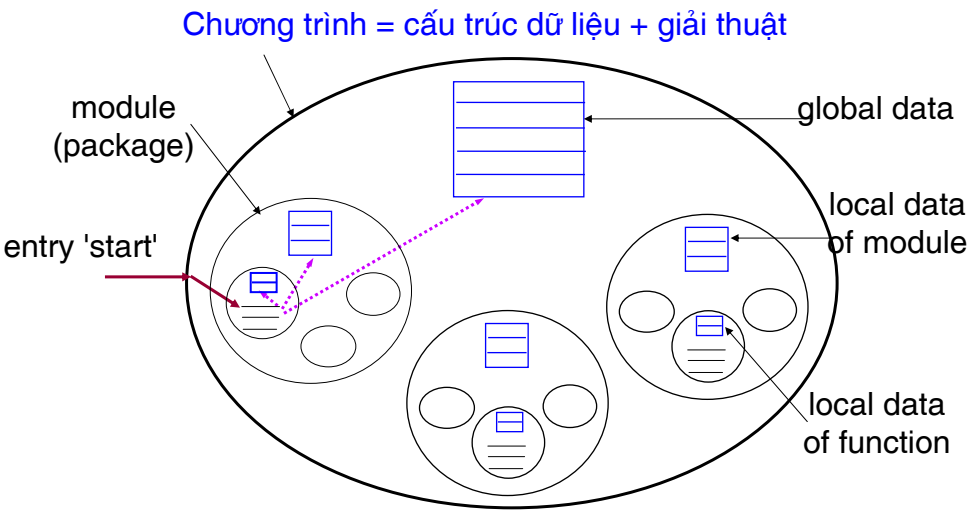
Bộ nhớ dây trễ, tĩnh điện. Giấy, phiếu đục lỗ. Băng từ

Bộ nhớ xuyến từ. Băng từ, trống từ, đĩa từ.

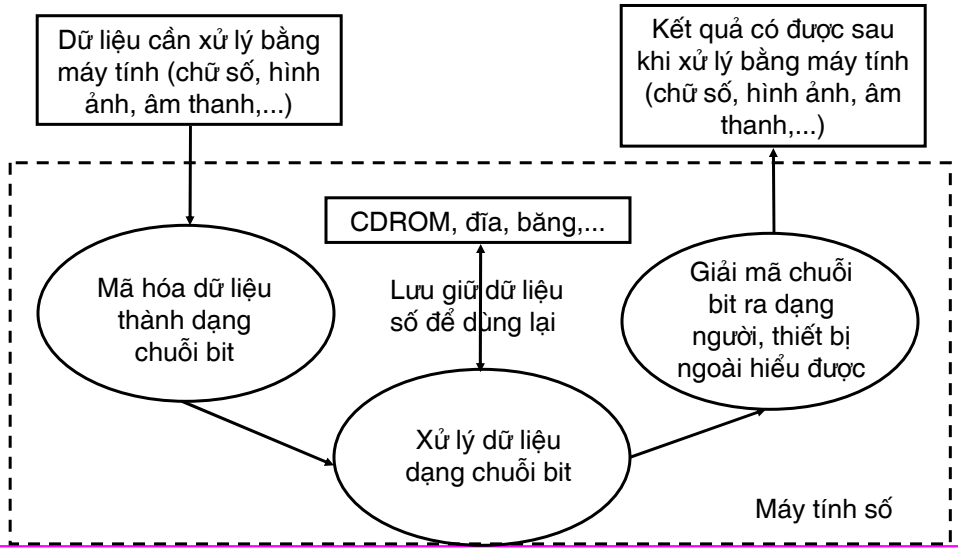
1.3 Dữ liệu & chương trình

- Các lệnh của chương trình (code) sẽ truy xuất (đọc và/hoặc ghi) thông tin (dữ liệu).
- Chương trình giải quyết bài toán nào đó có thể truy xuất nhiều dữ liệu khác nhau với tính chất rất đa dạng. Để truy xuất 1 dữ liệu cụ thể, ta cần 3 thông tin về dữ liệu đó :
  - **tên nhận dạng** (identifier) xác định vị trí của dữ liệu.
  - **kiểu dữ liệu** (type) miêu tả cấu trúc của dữ liệu.
  - **tầm vực truy xuất** (visibility) xác định các lệnh được phép truy xuất dữ liệu tương ứng.
- Chương trình cổ điển = dữ liệu + giải thuật.
- Chương trình con (function, subroutine,...) là 1 đoạn code thực hiện chức năng được dùng nhiều lần ở nhiều vị trí trong chương trình, được nhận dạng thông qua 1 tên gọi. Chương trình con cho phép cấu trúc chương trình, sử dụng lại code...

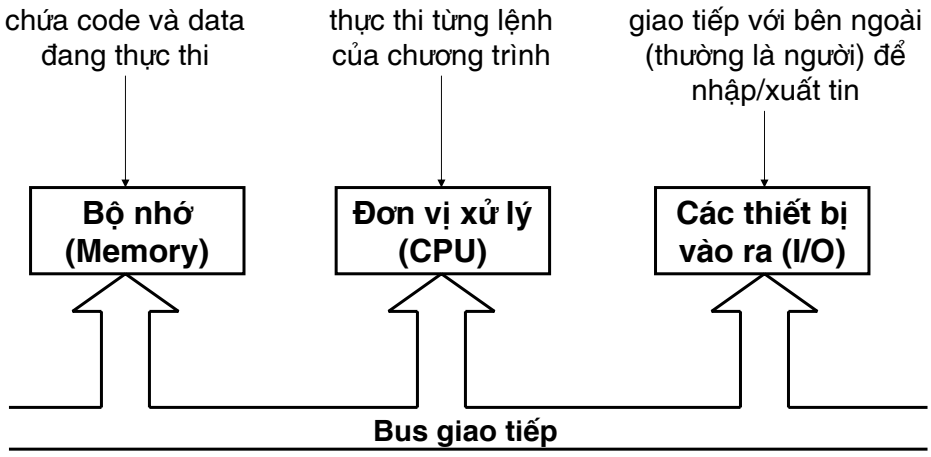
Cấu trúc 1 chương trình cổ điển



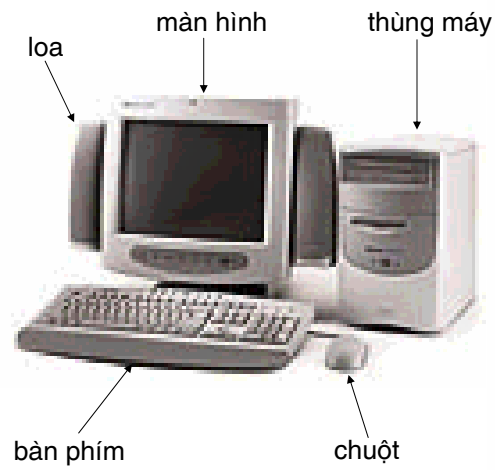
1.4 Qui trình tổng quát giải quyết bài toán bằng máy tính số

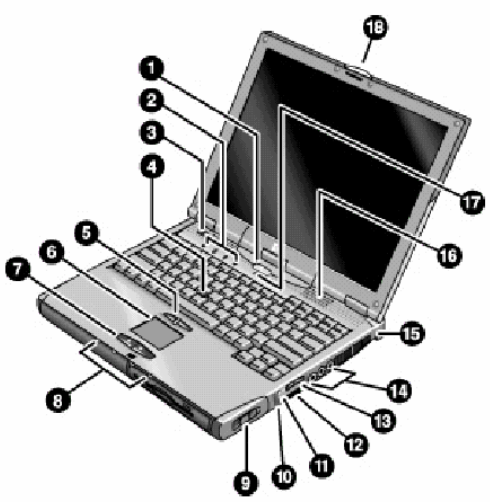


Mô hình máy tính số Von Neumann



Hình dạng vật lý của vài máy tính





1.5 Phương pháp phân tích từ-trên-xuống

Trong quá khứ, phương pháp thường sử dụng để phân tích bài toán là phương pháp từ-trên-xuống (top-down analysis).

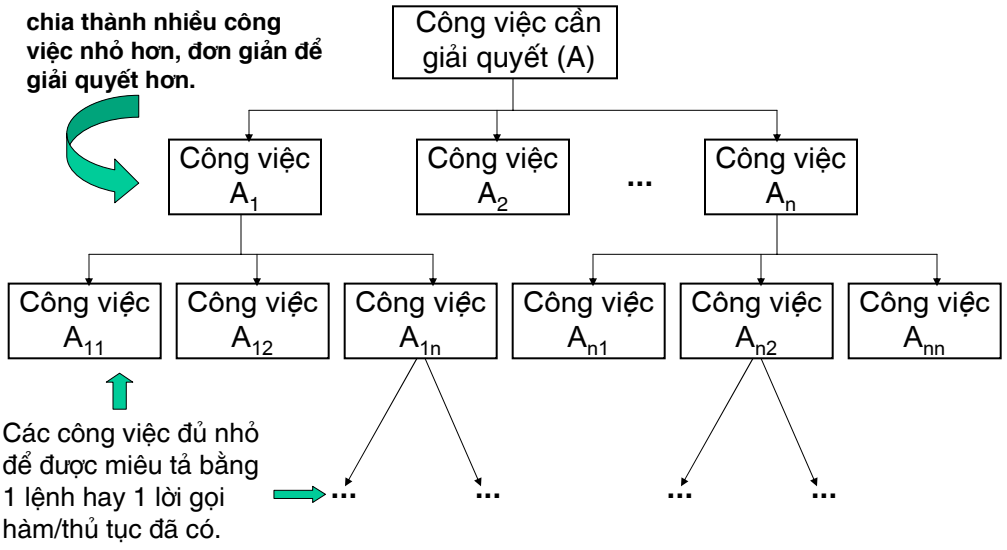
Nội dung của phương pháp này là xét xem, muốn giải quyết vấn đề nào đó thì cần phải làm những công việc nhỏ hơn nào. Mỗi công việc nhỏ hơn tìm được lại được phân thành những công việc nhỏ hơn nữa, cứ như vậy cho đến khi những công việc phải làm là những công việc thật đơn giản, có thể thực hiện dễ dàng.

Thí dụ việc học lấy bằng kỹ sư CNTT khoa CNTT ĐHBK TP.HCM có thể bao gồm 9 công việc nhỏ hơn là học từng học kỳ từ 1 tới 9, học học kỳ i là học n môn học của học kỳ đó, học 1 môn học là học m chương của môn đó,...

Hình vẽ của slide kế cho thấy trực quan của việc phân tích top-down.

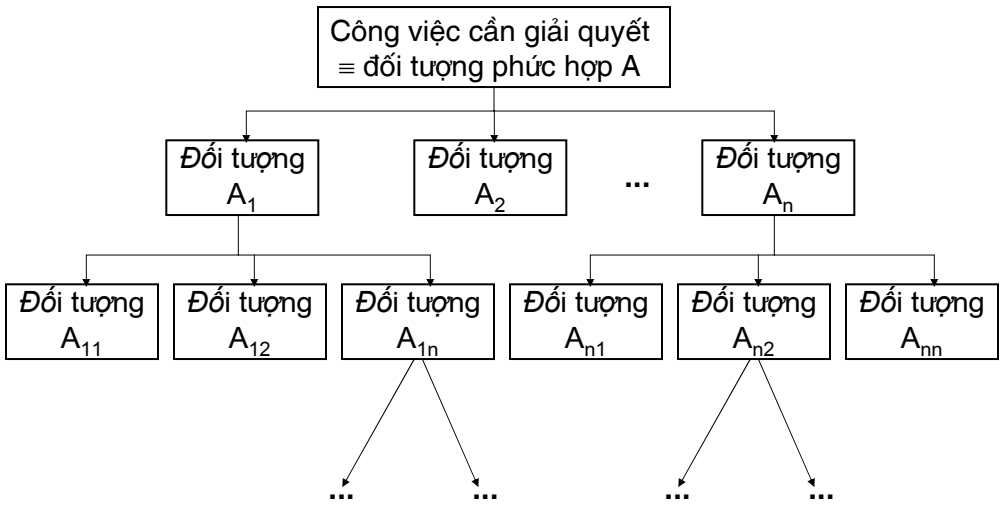
Phương pháp phân tích từ-trên-xuống

chia thành nhiều công việc nhỏ hơn, đơn giản để giải quyết hơn.



Các công việc đủ nhỏ để được miêu tả bằng 1 lệnh hay 1 lời gọi hàm/thủ tục đã có.

Phương pháp phân tích từ-trên-xuống



MÔN TIN HỌC

Chương 2

THỂ HIỆN DỮ LIỆU TRONG MÁY TÍNH SỐ

2.1 Cơ bản về việc lưu trữ và xử lý tin trong máy tính
2.2 Cơ bản về hệ thống số
2.3 Các phương pháp chuyển miêu tả số
2.4 Biểu diễn dữ liệu trong máy tính
2.5 Hệ thống file
2.6 Quản lý hệ thống file

2.1 Cơ bản về việc lưu trữ và xử lý tin trong máy tính

Phần tử nhớ nhỏ nhất của máy tính số chỉ có thể chứa 2 giá trị : 0 và 1 (ta gọi là bit).

Ta kết hợp nhiều phần tử nhớ để có thể miêu tả đại lượng lớn hơn. Thí dụ ta dùng 8 bit để miêu tả $2^8 = 256$ giá trị khác nhau. Dãy 8 bit nhớ được gọi là byte, đây là 1 ô nhớ trong bộ nhớ của máy tính.

Bộ nhớ trong của máy tính được dùng để chứa dữ liệu và code của chương trình đang thực thi. Nó là 1 dãy đồng nhất các ô nhớ 8 bit, mỗi ô nhớ được truy xuất độc lập thông qua địa chỉ của nó (tên nhận dạng). Thường ta dùng chỉ số từ 0 - n để miêu tả địa chỉ của từng ô nhớ.

Mặc dù ngoài đời ta đã quen dùng hệ thống số thập phân, nhưng về phần cứng bên trong máy tính, máy chỉ có thể chứa và xử lý trực tiếp dữ liệu ở dạng nhị phân. Do đó trong chương này, ta sẽ giới thiệu các khái niệm nền tảng về hệ thống số và cách miêu tả dữ liệu trong máy tính.

2.2 Cơ bản về hệ thống số

Hệ thống số (number system) là công cụ để biểu thị đại lượng. Một hệ thống số gồm 3 thành phần chính :

1. **cơ số** : số lượng ký số (ký hiệu để nhận dạng các số cơ bản).
2. **qui luật kết hợp các ký số** để miêu tả 1 đại lượng nào đó.
3. **các phép tính cơ bản** trên các số.

Trong 3 thành phần trên, chỉ có thành phần 1 là khác nhau giữa các hệ thống số, còn 2 thành phần 2 và 3 thì giống nhau giữa các hệ thống số.

Thí dụ :
- hệ thống số thập phân (**hệ thập phân**) dùng 10 ký số : 0,1,2,3,4,5,6,7,8,9.
- **hệ nhị phân** dùng 2 ký số : 0,1.
- hệ bát phân dùng 8 ký số : 0,1,2,3,4,5,6,7.
- **hệ thập lục phân** dùng 16 ký số : 0 đến 9,A,B,C,D,E,F.

Cơ bản về hệ thống số - Qui luật miêu tả lượng

Biểu diễn của lượng Q trong hệ thống số B (B>1) là :

$d_n d_{n-1} ... d_1 d_0 d_{-1} ... d_{-m} \Leftrightarrow$

$Q = d_n * B^n + d_{n-1} * B^{n-1} + ... + d_0 * B^0 + d_{-1} * B^{-1} + ... + d_{-m} * B^{-m}$

trong đó mỗi d_i là 1 ký số trong hệ thống B.

Trong thực tế lập trình bằng ngôn ngữ cấp cao, ta thường dùng hệ thống số thập phân để miêu tả dữ liệu số của chương trình (vì đã quen). Chỉ trong 1 số trường hợp đặc biệt, ta mới dùng hệ thống số thập lục phân (dạng ngắn của nhị phân) để miêu tả 1 vài giá trị nguyên, trong trường hợp này, qui luật biểu diễn của lượng nguyên Q trong hệ thống số B sẽ đơn giản là :

$d_n d_{n-1} ... d_1 d_0 \Leftrightarrow$

$Q = d_n * B^n + d_{n-1} * B^{n-1} + ... + d_1 * B^1 + d_0 * B^0$

trong đó mỗi d_i là 1 ký số trong hệ thống B.

Cơ bản về hệ thống số - Vài thí dụ

Thí dụ về biểu diễn các lượng trong các hệ thống số :

- lượng "mười bảy" được miêu tả là 17 trong hệ thập phân vì : $17 = 1*10^1+7*10^0$
- lượng "mười bảy" được miêu tả là 11 trong hệ thập lục phân vì : $11 = 1*16^1+1*16^0$
- lượng "mười bảy" được miêu tả là 10001 trong hệ nhị phân vì : $10001 = 1*2^4+0*2^3+0*2^2+0*2^1+1*2^0$

Trong môi trường sử dụng đồng thời nhiều hệ thống số, để tránh nhầm lẫn trong các biểu diễn của các lượng khác nhau, ta sẽ thêm ký tự nhận dạng hệ thống số được dùng trong biểu diễn liên quan. Thí dụ ta viết :

- 17_D để xác định sự biểu diễn trong hệ thống số thập phân.
- 11_H (hệ thống số thập lục phân.)
- 10001_B (hệ thống số nhị phân.)

2.3 Các phương pháp chuyển miêu tả số

Để chuyển 1 miêu tả số từ hệ thống số này sang hệ thống số khác, ta cần dùng 1 phương pháp chuyển thích hợp. Có 4 phương pháp sau tương ứng với từng yêu cầu chuyển tương ứng :

1. chuyển từ hệ thống số khác về thập phân.
2. chuyển từ nhị phân về thập lục phân (hay bát phân).
3. chuyển từ thập lục phân (hay bát phân) về nhị phân.
4. chuyển từ hệ thống số thập phân về hệ thống số khác.

Chuyển từ hệ thống khác về thập phân

Để chuyển 1 miêu tả số từ hệ thống số khác (nhị phân, thập lục phân hay bát phân) sang hệ thập phân, ta dùng công thức tính Q.

Thí dụ :

1. $1A2_H = 1*16^2+10*16^1+2*16^0 = 256+160+2 = 418_D$

2. $642_O = 6*8^2+4*8^1+2*8^0 = 384+32+2 = 418_D$

3. $110100010_B = 2^8 + 2^7+2^5+2^1 = 256+128+32+2 = 418_D$

Chuyển từ hệ thống nhị phân về thập lục phân

Lưu ý rằng có 1 mối quan hệ mật thiết giữa hệ nhị phân và thập lục phân (hay bát phân), đó là 4 ký số nhị phân tương đương với 1 ký số thập lục phân (hay 3 ký số nhị phân tương đương với 1 ký số bát phân) theo bảng tra sau :

Dec	Hex	Oct	Binary
0	0	00	0000
1	1	01	0001
2	2	02	0010
3	3	03	0011
4	4	04	0100
5	5	05	0101
6	6	06	0110
7	7	07	0111

Dec	Hex	Oct	Binary
8	8	10	1000
9	9	11	1001
10	A	12	1010
11	B	13	1011
12	C	14	1100
13	D	15	1101
14	E	16	1110
15	F	17	1111

Chuyển từ hệ thống nhị phân về thập lục phân

Để đổi 1 số nhị phân về thập lục phân (hay bát phân), ta đi từ phải sang trái và chia thành từng nhóm 4 ký số nhị phân (hay 3 ký số nhị phân), sau đó đổi từng nhóm 4 ký số (hay 3 ký số) thành 1 ký số thập lục phân tương đương (hay 1 ký số bát phân tương đương).

Thí dụ :

1. $110100010_B = 0001.1010.0010 = 1A2_H$
2. $110100010_B = 110.100.010 = 642_O$

Chuyển từ hệ thống thập lục phân về nhị phân

Để đổi 1 số thập lục phân (hay bát phân) về nhị phân, ta đổi từng ký số thập lục phân (hay bát phân) thành từng nhóm 4 ký số nhị phân (hay 3 ký số nhị phân).

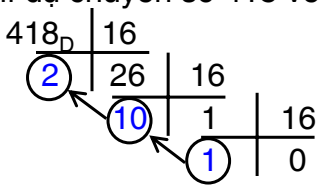
Thí dụ :

1. $1A2_H = 0001.1010.0010 = 110100010_B$
2. $642_O = 110.100.010 = 110100010_B$

Chuyển từ hệ thống thập phân về hệ thống khác

Để đổi 1 số thập phân về hệ thống số khác, ta hãy chia số cần đổi cho cơ số đích để có được thương và dư số, ta **lặp lại** hoạt động chia thương số cho cơ số đích để có được thương và dư số mới, cứ thế lặp mãi cho đến khi thương số = 0 thì dừng lại. Ghép các dư số theo chiều ngược chiều lặp để tạo ra kết quả (đó là sự miêu tả số tương đương nhưng ở hệ thống số khác).

Thí dụ chuyển số 418 về miêu tả tương ứng trong hệ thập lục :



Kết quả là $418_D = 1A2_H$

Chuyển từ hệ thống thập lục phân về bát phân

Để đổi 1 số thập lục phân về bát phân (hay ngược lại), ta nên chuyển tuần tự từ thập lục phân về nhị phân, rồi từ nhị phân về bát phân.

Cơ bản về hệ thống số - Các phép tính

Các phép tính cơ bản trong 1 hệ thống số là :

1. phép cộng (+).
2. phép trừ (-).
3. phép chia (/).
4. phép nhân (*).
5. phép dịch trái n ký số (<< n).
6. phép dịch phải n ký số (>> n).

Ngoài ra do đặc điểm của hệ nhị phân, hệ này còn cung cấp 1 số phép tính sau (các phép tính luận lý) :

1. phép OR bit (|).
2. phép AND bit (&).
3. phép XOR bit (^).
4. ....

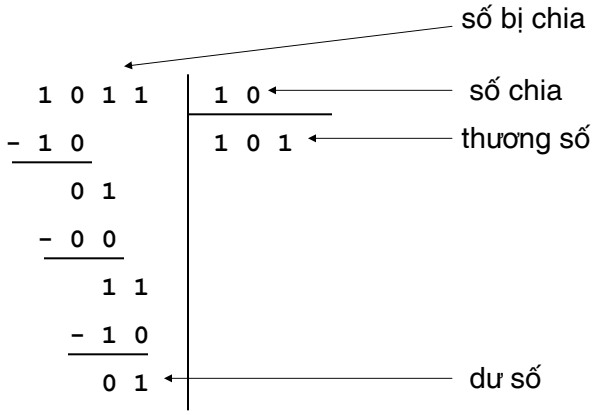
Thí dụ về phép cộng, trừ, nhân

Thí dụ về các phép tính cơ bản (các giá trị đều được biểu diễn bằng hệ nhị phân :

```
  0 1 1 0        1 0 0 1          1 0 0 1
+ 0 0 1 1      - 0 0 1 1        * 0 1 0 1
---------      ---------        ---------
  1 0 0 1        0 1 1 0          1 0 0 1
                                0 0 0 0
                              1 0 0 1
                              ---------
                              1 0 1 1 0 1
```

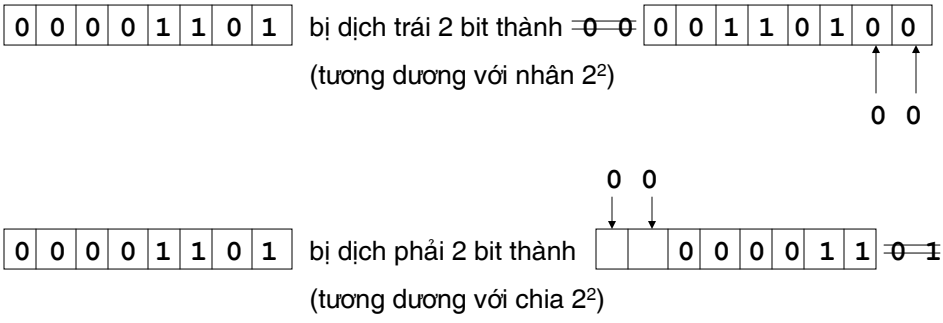
Thí dụ về phép chia

Thí dụ về các phép tính cơ bản (các giá trị đều được biểu diễn bằng hệ nhị phân) :



Thí dụ về phép dịch ký số

Thí dụ về các phép tính dịch ký số (các giá trị đều được biểu diễn bằng hệ nhị phân) :



Các phép tính của đại số Boole

Đại số Boole nghiên cứu các phép toán thực hiện trên các biến chỉ có 2 giá trị 0 và 1, tương ứng với hai thái cực luận lý "sai" và "đúng" (hay "không" và "có") của đời thường. Các phép toán này gồm :

x	y	**not** x	x **and** y	x **nand** y	x **or** y	x **nor** y	x **xor** y
0	0	1	0	1	0	1	0
0	1		0	1	1	0	1
1	0	0	0	1	1	0	1
1	1		1	0	1	0	0

Biểu thức Boole là 1 biểu thức toán hoc cấu thành từ các phép toán Boole trên các toán hạng là các biến chỉ chứa 2 trị 0 và 1.

Hàm Boole

Một hàm Boole theo n biến boole (hàm n ngôi) là 1 biểu thức boole cấu thành từ các phép toán Boole trên các biến boole.

Thay vì miêu tả hàm boole bằng biểu thức boole, ta có thể miêu tả hàm boole bằng bảng thực trị. Bảng thực trị của hàm boole n biến có 2^n hàng, mỗi hàng miêu tả 1 tổ hợp trị cụ thể của các biến và giá trị cụ thể của hàm tương ứng với tổ hợp trị này (xem slide ngay trước).

Như vậy 1 hàm boole n biến được miêu tả như 1 chuỗi 2^n bit $\Rightarrow$ có chính xác 2^{2^n} hàm boole n ngôi khác nhau. Cụ thể có :

$2^{2^1} = 4$ hàm boole 1 ngôi khac nhau

$2^{2^2} = 2^4 = 16$ hàm boole 2 ngôi khac nhau

$2^{2^3} = 2^8 = 256$ hàm boole 3 ngôi khac nhau

Các đơn vị nhớ thường dùng

Máy tính dùng trực tiếp hệ nhị phân, các đơn vị biểu diễn thông tin thường dùng là :

1. **bit :** miêu tả 2 giá trị khác nhau (đúng/sai, 0/1,..)
2. **byte** : 8bit, có thể miêu tả được $2^8 = 256$ giá trị khác nhau.
3. **word :** 2 byte, có thể miêu tả được $2^{16} = 65536$ giá trị khác nhau.
4. **double word** : 4 byte, có thể miêu tả được $2^{32} =$ 4.294.967.296 giá trị khác nhau.
5. **KB (kilo byte)** = 2^{10} = 1024 byte.
6. **MB (mega byte)** = 2^{20} = 1024KB = 1.048.576 byte.
7. **GB (giga byte)** = 2^{30} = 1024MB = 1.073.741.824 byte.
8. **TB (tetra byte)** = 2^{40} = 1024GB = 1.099.511.627.776 byte.

Thí dụ, RAM của máy bạn là 512MB, đĩa cứng là 300GB.

2.4 Biểu diễn số nguyên trong Visual Basic

Tùy ngôn ngữ lập trình mà cách biểu diễn số trong máy có những khác biệt nhất định. Riêng VB có nhiều phương pháp biểu diễn số khác nhau, trong đó 2 cách thường dùng là số nguyên và số thực.

Máy dùng 1 word (2 byte) để chứa dữ liệu nguyên (Integer) theo qui định cụ thể ở slide sau.

Vì mỗi ô nhớ máy tính chỉ chứa được 1 byte, do đó ta phải dùng nhiều ô liên tiếp (2 hay 4) để chứa số nguyên. Có 2 cách chứa các byte của số nguyên (hay dữ liệu khác) vào các ô nhớ : BE & LE.

Cách BE (Big Endian) chứa byte trọng số cao nhất vào ô nhớ địa chỉ thấp trước, sau đó lần lượt đến các byte còn lại. Cách LE (Little Endian) chứa byte trong số nhỏ nhất vào ô nhớ địa chỉ thấp trước, sau đó lần lượt đến các byte còn lại.

VB sử dụng cách LE để chứa số nguyên vào bộ nhớ (Integer và Long).

Biểu diễn số nguyên trong Visual Basic

- Phần dương có 32768 số từ số 0 tới 32767, được miêu tả theo công thức Q.
- Phần âm có 32768 số từ -32768 tới -1, được miêu tả ở dạng số bù 2 như sau :
- Số bù 1 của 1 số n bit là n bit mà mỗi bit là ngược với bit gốc (0 → 1 và 1 → 0)
- Số bù 2 của 1 số n bit là số bù 1 của số đó rồi tăng lên 1 đơn vị.

Sự biểu diễn	giá trị
00000000 00000000	0
00000000 00000001	1
....	
01111111 11111111	32767
10000000 00000000	-32768
10000000 00000001	-32767
....	
11111111 11111111	-1

Biểu diễn số nguyên trong VB - Thí dụ

- Số 15 được miêu tả dưới dạng nhị phân 16 bit như sau :
  0000 0000 0000 1111
- Do đó, nếu dùng kiểu Integer để lưu số 15, ta dùng 16 bit như trên hay viết ngắn gọn là $000F_H$. Nếu lưu vào bộ nhớ dạng LE thì ô nhớ có địa chỉ thấp (i) chứa byte $0F_H$, và ô nhớ kế (i+1) chứa byte 00. Nếu dùng kiểu Long để lưu số 15, ta dùng 4 byte $0000000F_H$ và lưu vào bộ nhớ dạng LE tốn 4 ô nhớ với giá trị lần lượt từ địa chỉ thấp đến cao là $0F_H$, 00, 00, 00.
- Số bù 1 của 15 là 1111 1111 1111 0000, số bù 2 của 15 là
  1111 1111 1111 0001
- Như vậy -15 được lưu vào máy dạng Integer là 2 byte có giá trị $FFF1_H$. Nếu lưu vào ô nhớ dạng LE thì ô nhớ có địa chỉ thấp (i) chứa byte $F1_H$, và ô nhớ kế (i+1) chứa byte FF_H.

Biểu diễn số thực trong Visual Basic

Để miêu tả được các giá trị nguyên nằm ngoài phạm vi từ -32768 đến 32767, VB cung cấp kiểu 'Long', kiểu này dùng 4 byte để miêu tả 1 giá trị nguyên với cùng nguyên tắc như kiểu Integer. Kết quả là kiểu Long miêu tả các số nguyên trong phạm vi từ -2 tỉ đến 2 tỉ.

Trong khoa học, ta có thể miêu tả số thực theo dạng $\pm m*B^e$, m gọi là định trị, B là cơ số và e là số mũ. Như vậy 1 số thực cụ thể có thể được miêu tả bởi rất nhiều miêu tả khác nhau, trong đó miêu tả có $0.1 \le m < 1$ được gọi là miêu tả chính tắc của số thực. Đây là miêu tả mà máy tính sẽ dùng.

$\pm m \times B^e$

- **m (mantissa) quyết *định độ* chính xác**
- **B (base)**
- **e (exponent) quyết *định độ* lớn/nhỏ**

Biểu diễn số thực trong Visual Basic

Trước khi lưu vào máy tính, số thực được đổi về dạng miêu tả nhị phân dưới dạng ±1.m*2e (m là chuỗi bit nhị phân miêu tả phần lẻ). VB lưu số thực theo chuẩn IEEE 754, dùng 1 trong 2 dạng lưu :

- Chính xác đơn (Single) : VB dùng 4 byte - 4 ô nhớ (32 bit) để lưu số thực theo dạng thức cụ thể sau :

trong đó bit S = 1 (âm), =0 (dương).

M = m & E = 127 + e

- Chính xác kép (Double) : VB dùng 8 byte - 8 ô nhớ (64 bit) để lưu số thực theo dạng thức cụ thể sau :

trong đó bit S = 1 (âm), =0 (dương); M = m & E = 1023 + e

Biểu diễn số thực trong VB - Thí dụ

Thí dụ giá trị -1.5 được miêu tả dạng nhị phân là $-1.1*2^0$.

- Do đó nếu dùng kiểu Single chứa số thực -1.5, ta tốn 4 byte (32 bit) với các thành phần S = 1, M = 10...0 (22 bit 0), E = 127. Kết quả, giá trị của 4 byte miêu tả số -1.5 như sau : BF C0 00 00
- Tương tự, nếu dùng kiểu Double chứa số thực -1.5, ta tốn 8 byte (64 bit) với các thành phần S = 1, M = 10...0 (51 bit 0), E = 1023. Kết quả, giá trị của 8 byte miêu tả số -1.5 như sau : BF F8 00 00 00 00 00 00.
- VB dùng cách chứa LE, do đó giá trị -1.5 được lưu vào bộ nhớ theo kiểu Single sẽ chiếm 4 byte theo giá trị lần lượt từ địa chỉ thấp đến cao là 00 00 C0 BF. Tương tự nếu miêu tả -1.5 vào bộ nhớ theo kiểu Double thì sẽ cần 8 ô nhớ với giá trị lần lượt **từ địa chỉ thấp đến cao** là 00 00 00 00 00 00 F8 BF.

Biểu diễn chuỗi ký tự trong Visual Basic

Chuỗi ký tự là danh sách nhiều ký tự, mỗi ký tự được miêu tả trong máy bởi n bit nhớ :

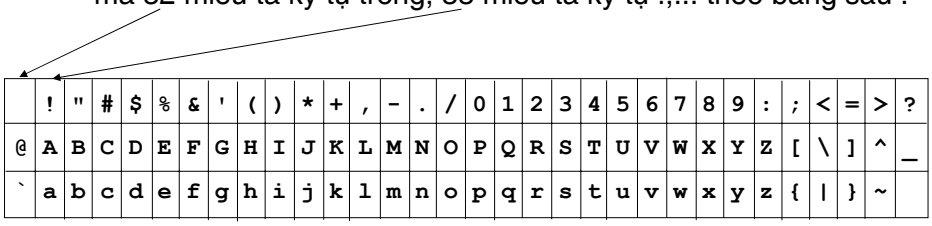
- mã ASCII dùng 7 bit (dùng luôn 1 byte nhưng bỏ bit 8) để miêu tả 1 ký tự ⇒ tập ký tự mà mã ASCII miêu tả được là 128.
- mã ISO8859-1 dùng 8 bit (1 byte) để miêu tả 1 ký tự ⇒ tập ký tự mà mã ISO8859-1 miêu tả được là 256.
- mã Unicode trên Windows dùng 16 bit (2 byte) để miêu tả 1 ký tự ⇒ tập ký tự mà mã Unicode trên Windows miêu tả được là 65536.
- ...

Hiện có nhiều loại mã tiếng Việt khác nhau, đa số dùng mã ISO8859-1 rồi qui định lại cách hiển thị 1 số ký tự thành ký tự Việt. Riêng Unicode là bộ mã thống nhất toàn cầu, trong đó có đủ các ký tự Việt.

Bảng mã ASCII 7 bit

Mã ASCII dùng các giá trị (mã) từ 0 - 127 để miêu tả các ký tự :

- mã từ 0 - 31 là các mã điều khiển như CR=13 (Carriage Return), LF=10 (Line Feed), ESC=27 (Escape)...
- mã 32 miêu tả ký tự trống, 33 miêu tả ký tự !,... theo bảng sau :

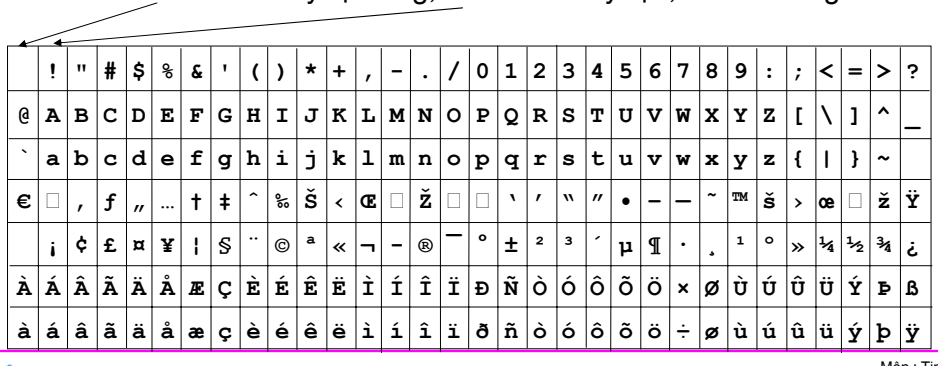


	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

Bảng mã ISO8859-1 (8 bit)

Mã ISO8859-1 dùng các giá trị (mã) từ 0 - 255 để miêu tả các ký tự (128 mã ký tự đầu qui định giống như mã ASCII) :

- mã từ 0 - 31 là các mã điều khiển như CR=13 (Carriage Return), LF=10 (Line Feed), ESC=27 (Escape)...
- mã 32 miêu tả ký tự trống, 33 miêu tả ký tự !,... theo bảng sau :



	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
€	□	‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ	□	Ž	□	□	‘	’	“	”	•	–	—	˜	™	š	›	œ	□	ž	Ÿ
	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Bảng mã tiếng Việt ĐHBK 1 byte

Mã ĐHBK 1 byte có được bằng cách hiệu chỉnh bảng mã ISO8859-1 :

- mã từ 0 - 31 là các mã điều khiển như CR=13 (Carriage Return), LF=10 (Line Feed), ESC=27 (Escape)...
- mã 32 miêu tả ký tự trống, 33 miêu tả ký tự !,... theo bảng sau :

	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	Ỳ	_
Ỷ	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	Ý	Ỹ	Đ	Ấ	
Á	À	Ả	Ã	Ạ	É	È	Ẻ	Ẽ	Ẹ	Í	Ì	Ỉ	Ĩ	Ị	Ó	Ò	Ỏ	Õ	Ọ	Ú	Ù	Ủ	Ũ	Ụ	Ă	Ắ	Ằ	Ẳ	Ẵ	Ặ	Â
Ấ	Ầ	Ẩ	Ẫ	Ậ	Ê	Ế	Ề	Ể	Ễ	Ệ	Ô	Ố	Ồ	Ổ	Ỗ	Ộ	Ơ	Ớ	Ờ	Ở	Ỡ	Ợ	.	Ứ	Ừ	Ử	Ữ	Ự	đ	á	à
ả	ã	ạ	é	è	ẻ	ẽ	ẹ	í	ì	ỉ	ĩ	ị	ó	ò	ỏ	õ	ọ	ú	ù	ủ	ũ	ụ	ă	ắ	ằ	ẳ	ẵ	ặ	â	ấ	ầ
ẩ	ẫ	ậ	ê	ế	ề	ể	ễ	ệ	ô	ố	ồ	ổ	ỗ	ộ	ơ	ớ	ờ	ở	ỡ	ợ	ư	ứ	ừ	ử	ữ	ự	ý	ỳ	ỷ	ỹ	ỵ

Một phần mã tiếng Việt Unicode

Mã Unicode Windows dùng 2 byte để miêu tả 1 ký tự :

- 256 mã đầu từ 0 - 255 giống y như mã ISO8859-1.
- mã từ 256 trở đi chứa các ký tự của hầu hết các ngôn ngữ trên thế giới (quá khứ, hiện tại và tương lai).
- thí dụ sau là 1 phần mã tiếng Việt trong mã Unicode :

mã $1ea0_H$ biểu diễn ký tự Ạ

mã $1ef9_H$ biểu diễn ký tự ỹ

Ạ	ạ	Ả	ả	Ấ	ấ	Ầ	ầ	Ẩ	ẩ	Ẫ	ẫ	Ậ	ậ	Ắ	ắ	Ằ	ằ	Ẳ	ẳ	Ẵ	ẵ	Ặ	ặ	Ẹ	ẹ	Ẻ	ẻ	Ẽ	ẽ	Ế	ế
Ề	ề	Ể	ể	Ễ	ễ	Ệ	ệ	Ỉ	ỉ	Ị	ị	Ọ	ọ	Ỏ	ỏ	Ố	ố	Ồ	ồ	Ổ	ổ	Ỗ	ỗ	Ộ	ộ	Ớ	ớ	Ờ	ờ	Ở	ở
Ỡ	ỡ	Ợ	ợ	Ụ	ụ	Ủ	ủ	Ứ	ứ	Ừ	ừ	Ử	ử	Ữ	ữ	Ự	ự	Ỳ	ỳ	Ỵ	ỵ	Ỷ	ỷ	Ỹ	ỹ						

Mã hóa dữ liệu của ứng dụng

Số nguyên (Integer, Long), số thực (Single, Double), chuỗi ký tự (String) là những dạng mã hóa dữ liệu phổ dụng, ngoài ra mỗi ứng dụng có thể cần có cách mã hóa riêng để mã hóa dữ liệu đặc thù của mình như hình ảnh, âm thanh,...

Trong chương 5 và 6 chúng ta sẽ trình bày chi tiết các kiểu dữ liệu mà ngôn ngữ VB hỗ trợ.

Nhưng ta đã trình bày trong slide 15 (chương 1), dù dùng cách mã hóa cụ thể nào thì kết quả của việc mã hóa phải là 1 chuỗi bit (hay chuỗi byte) để có thể được lưu trữ và xử lý bên trong máy tính.

Bộ nhớ của máy tính thường có dung lượng không lớn nên ta chỉ dùng nó để chứa code và dữ liệu của chương trình đang thực thi.

1 máy tính có thể lưu trữ rất nhiều chương trình và dữ liệu của chúng trên các thiết bị chứa tin (bộ nhớ ngoài) như đĩa mềm, đĩa cứng, CDROM,...

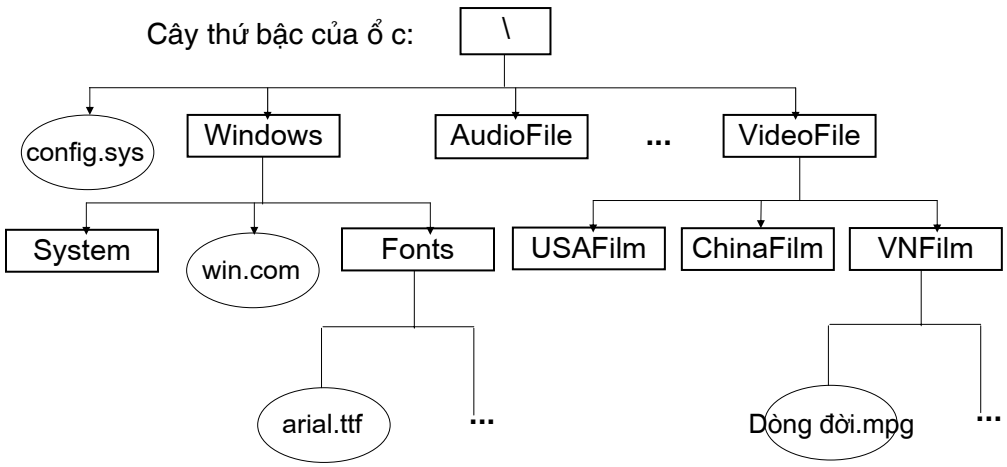
2.5 Hệ thống file

- code của 1 chương trình, chuỗi byte miêu tả dữ liệu được lưu trữ trên thiết bị chứa tin trong 1 phần tử chứa tin luận lý được gọi là file.
- 1 thiết bị chứa tin thường chứa rất nhiều file. Để nhận dạng và truy xuất 1 file, ta dùng tên nhận dạng gán cho mỗi file. Để dễ dùng file, tên nhận dạng của nó sẽ ở dạng tên gợi nhớ (chuỗi ký tự miêu tả ngữ nghĩa của nội dung file), thí dụ như file "luận án tốt nghiệp.doc" chứa toàn bộ nội dung luận án tốt nghiệp của người dùng máy.
- Nếu ta dùng không gian phẳng để đặt tên cho các file trên 1 thiết bị chứa tin thì vì số lượng file quá lớn nên ta khó lòng đặt tên, nhận dạng, xử lý,... (nói chung là quản lý) từng file.
- Để giải quyết vấn đề trên ta dùng không gian cây thứ bậc để tổ chức và quản lý các file trên từng thiết bị chứa tin.

Thiết bị chứa tin : Không gian cây thứ bậc

- Để tạo không gian cây thứ bậc, ta dùng khái niệm thư mục (directory).
- thư mục là phần tử chứa nhiều phần tử bên trong nó : có thể là file hay thư mục. Thường ta sẽ dùng thư mục để chứa những phần tử con có mối quan hệ mật thiết nào đó, thí dụ như thư mục chứa các ảnh kỷ niệm, thư mục chứa các file nhạc ưa thích,...
- Thiết bị chứa tin vật lý (đĩa mềm, đĩa cứng, CDROM,...) được trừu tượng hóa như là 1 thư mục (ta gọi thư mục đặc biệt này là thư mục gốc). Thư mục gốc chứa nhiều phần tử con bên trong, mỗi phần tử con của thư mục gốc thường là thư mục con nhưng cũng có thể là file. Mỗi thư mục con lại có thể chứa nhiều thư mục con hay file... và cứ thế ta sẽ hình thành 1 cây thứ bậc các thư mục và file.
- Ta cũng dùng tên gợi nhớ để nhận dạng từng thư mục. Trong không gian cây thứ bậc, ta sẽ dùng khái niệm đường dẫn (pathname) để nhận dạng 1 file hay 1 thư mục.

Thí dụ về hệ thống file



Đường dẫn tuyệt đối và tương đối

- Đường dẫn (pathname) là thông tin để tìm kiếm (xác định) 1 phần tử từ 1 vị trí nào đó, nó chứa danh sách chính xác các tên gọi nhớ của các phần tử mà ta phải đi qua xuất phát từ vị trí đầu để đến phần tử cần tìm.
- ta dùng 1 dấu ngăn đặc biệt để ngăn cách 2 tên gọi nhớ liên tiếp nhau trong đường dẫn (trong Windows, dấu ngăn là '\')
- Tên thư mục gốc luôn là '\'.
- Có 2 khái niệm đường dẫn : đường dẫn tuyệt đối và đường dẫn tương đối. Đường dẫn tuyệt đối là đường dẫn xuất phát từ thư mục gốc, đường dẫn tương đối xuất phát từ thư mục làm việc (working directory).
- Trước khi **ứng dụng** bắt đầu chạy, hệ thống sẽ khởi động thư mục làm việc cho ứng dụng (theo cơ chế nào đó). Trong quá trình thực thi, ứng dụng có quyền thay đổi thư mục làm việc theo yêu cầu riêng.

Đường dẫn tuyệt đối và tương đối (tt)

- Xét cây thứ bậc của ổ c: trên slide 54, đường dẫn tuyệt đối sau sẽ nhận dạng chính xác file arial.ttf trong thư mục 'Fonts' :

  c:\Windows\Fonts\arial.ttf
- Nếu thư mục working của chương trình hiện là c:\Windows\Fonts thì ta có thể dùng đường dẫn tương đối sau đây để xác định file arial.ttf :

  arial.ttf
- Đường dẫn tuyệt đối thường dài hơn đường dẫn tương đối nhưng nó luôn có giá trị bất chấp ứng dụng đang ở thư mục working nào.
- Đường dẫn tương đối thường gọn hơn (đa số chỉ chứa tên file cần truy xuất vì ứng dụng sẽ thiết lập thư mục working là thư mục chứa các file mà ứng dụng truy xuất) nhưng chỉ có giá trị với 1 thư mục working cụ thể.
- Trong 1 vài trường hợp đặc biệt, ta phải dùng đường dẫn tương đối ngay cả nó dài và phức tạp hơn đường dẫn tuyệt đối.

2.6 Quản lý hệ thống file

- Hình dạng và cấu trúc của 1 hệ thống file của 1 thiết bị chứa tin sẽ do người dùng thiết lập nhờ các tác vụ phổ biến như : tạo/xóa thư mục, tạo/xóa file, copy/move file/thư mục từ nơi này đến nơi khác.
- Nhưng trước khi thực hiện 1 tác vụ nào đó, người dùng thường duyệt file : làm hiển thị cấu trúc của hệ thống file ở 1 dạng nào đó để quan sát nó dễ dàng.
- Hệ thống dùng nhiều cơ chế khác nhau để bảo vệ việc truy xuất file bởi người dùng. 1 trong các cơ chế mà Windows XP dùng là kết hợp với mỗi file 1 số thuộc tính truy xuất, mỗi thuộc tính được lưu trữ trong 1 bit :
  - Read Only, nếu = 1 thì hệ thống không cho các ứng dụng xóa/hiệu chỉnh phần tử.
  - Hidden, nếu = 1 thì hệ thống sẽ dấu không hiển thị phần tử bởi các ứng dụng duyệt file.
  - Archive được thiết lập =1 nếu phần tử bị hiệu chỉnh nội dung (phục vụ cho cơ chế backup tăng dần).

Tiện ích quản lý hệ thống file

- Tất cả tác vụ liên quan đến hệ thống file được gọi là tác vụ quản lý hệ thống file.
- hệ thống sẽ cung cấp 1 ứng dụng (tiện ích) để người dùng dễ dàng thực hiện các tác vụ quản lý file. Thí dụ trên Windows ta thường dùng tiện ích "Windows Explorer" để quản lý hệ thống file.
- Có 4 cách phổ biến để chạy 1 ứng dụng (tiện ích) :
  1. double-click vào icon miêu tả ứng dụng trên màn hình desktop (phải tạo icon shortcut chương trình trước khi dùng cách chạy này).
  2. duyệt và chọn ứng dụng từ menu Start.Programs...
  3. chạy trình Windows Explorer (từ menu Start.Programs.Accessories.Windows Explorer), duyệt thư mục tìm file ứng dụng, ấn kép chuột vào file để chạy nó.
  4. vào menu Start.Run, rồi nhập hàng lệnh chứa đường dẫn xác định file chương trình và các tham số hàng lệnh.

Cửa sổ của WE & các phần tử giao diện chính

MenuBar chứa tất cả tác vụ mà ứng dụng hỗ trợ

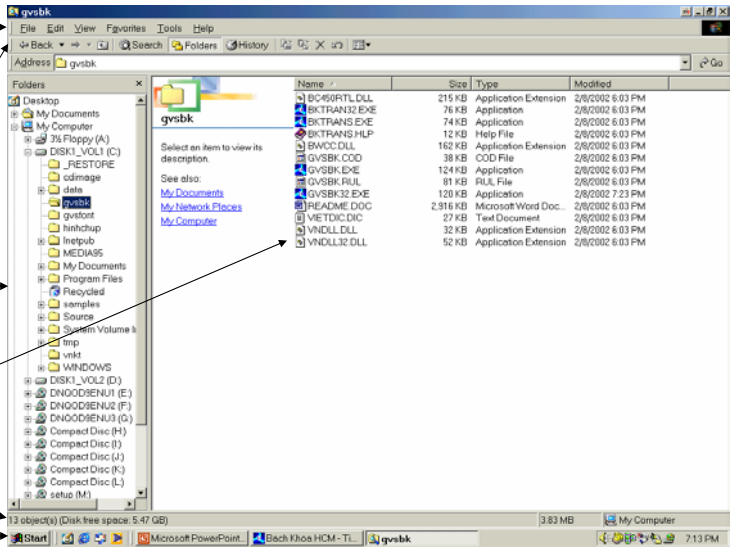
Toolbar chứa các icon tác vụ thường dùng

TreeCtrl hiển thị hệ thống file dạng cây

ListCtrl hiển thị các phần tử trong thư mục

StatusBar

Taskbar



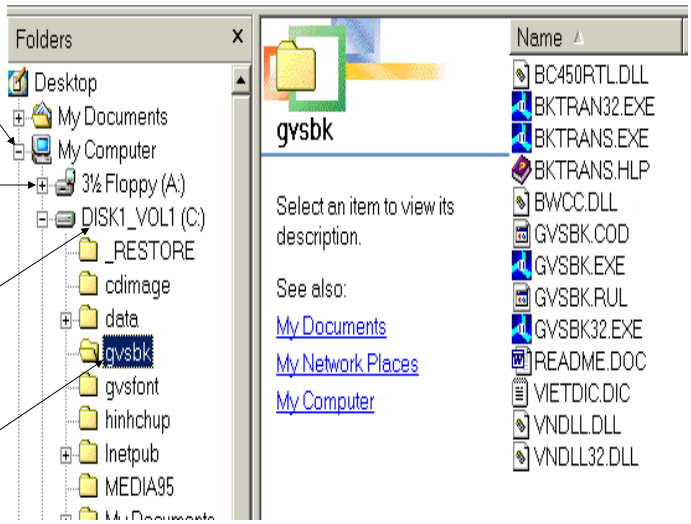
Các thao tác duyệt hệ thống file

Click vào ô - để thu nhỏ nội dung thư mục

Click vào ô + để chi tiết hóa nội dung thư mục.

Nhưng tốt nhất là double-click vào tên thư mục để chi tiết hóa/thu nhỏ nội dung

Click vào tên thư mục để hiển thị nội dung chi tiết của nó



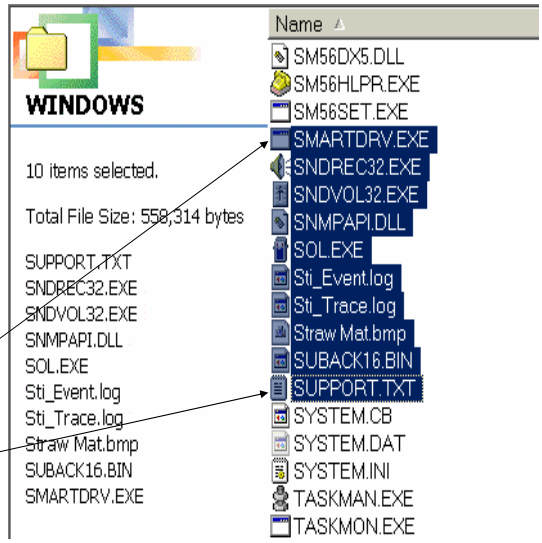
Các tác vụ xử lý file

Qui trình chung để thực hiện tác vụ trên 1 hay nhiều phần tử nào đó là :

1. chọn 1 hay nhiều phần tử cần xử lý.
2. chọn option trong menu hay icon trong toolbar thực hiện tác vụ mong muốn.

Chọn nhiều phần tử liên tiếp :

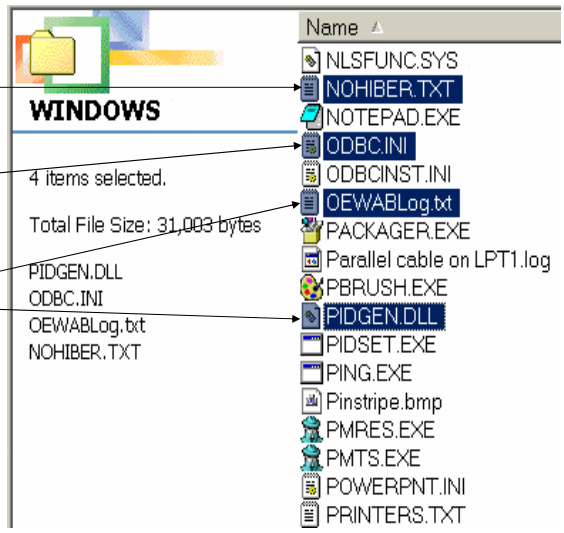
1. chick vào phần tử đầu,
2. ấn và giử phím Shift,
3. click vào phần tử cuối.
4. thả phím Shift.



Các tác vụ xử lý file

Chọn nhiều phần tử rời rạc :

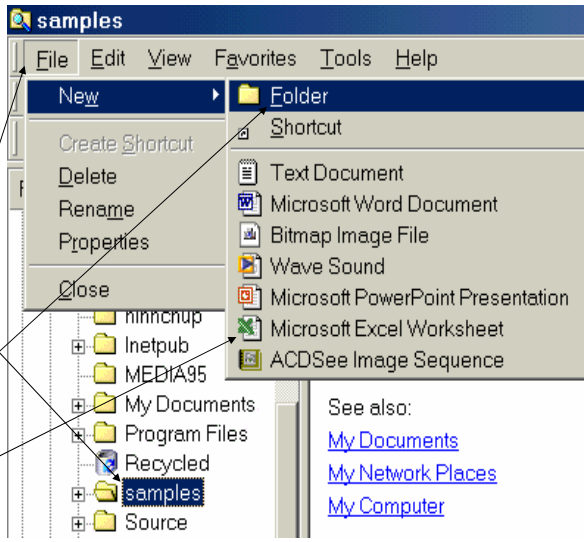
1. chick vào phần tử đầu,
2. ấn và giử phím Ctrl,
3. dời mouse đến từng phần tử cần chọn rồi click vào nó.
4. lặp lại bước 3 nhiều lần cho nhiều phần tử
5. thả phím Ctrl.



Tạo thư mục/file mới

Thường việc tạo file mới được thực hiện bên trong ứng dụng. Qui trình tạo mới 1 thư mục/file trong WE như sau :

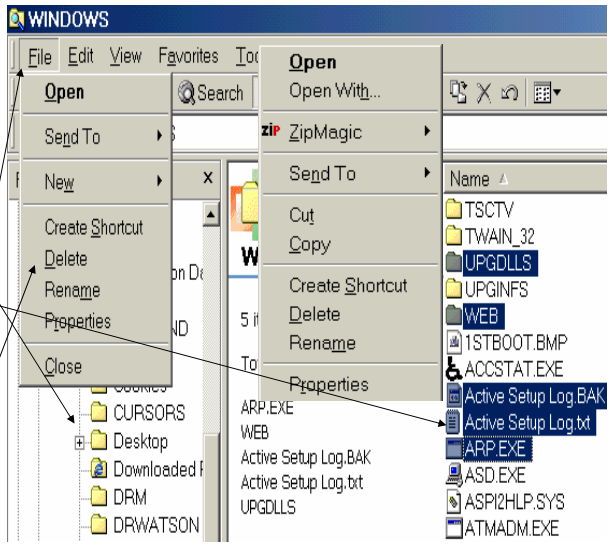
1. duyệt cây thư mục trong cửa sổ bên trái và chọn thư mục mà ở đó bạn muốn tạo thư mục/file mới.
2. chọn menu File.New
3. nếu muốn tạo thư mục, chọn Folder.
4. nếu muốn tạo file, chọn loại file trong danh sách.



Xóa thư mục/file đang tồn tại

Qui trình xóa 1 thư mục/file trong WE như sau :

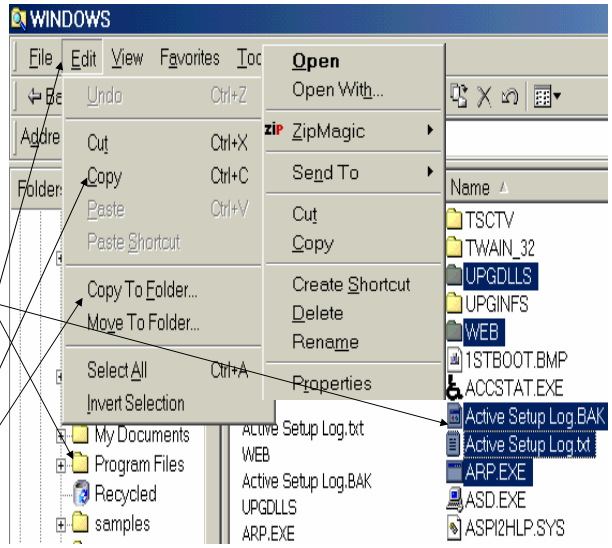
1. duyệt cây thư mục trong cửa sổ bên trái và chọn thư mục mà ở đó bạn muốn xóa thư mục/file.
2. chọn các phần tử cần xóa trong ListCtrl bên phải.
3. chọn menu File hay ấn phải chuột vào vị trí chọn các phần tử để hiển thị menu các tác vụ có thể thực hiện.
4. chọn option "Delete"



Copy thư mục/file vào clipboard

Qui trình copy thư mục/file trong WE như sau :

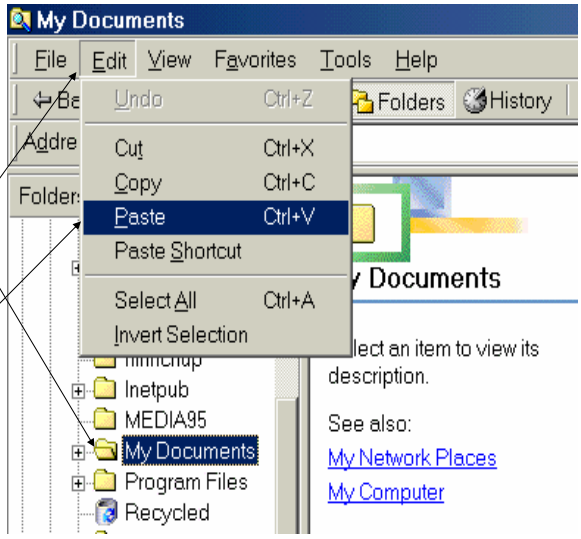
1. duyệt cây thư mục trong cửa sổ bên trái và chọn thư mục mà ở đó bạn muốn copy thư mục/file.
2. chọn các phần tử cần copy trong ListCtrl bên phải.
3. chọn menu Edit hay ấn phải chuột vào vị trí chọn các phần tử để hiển thị menu các tác vụ có thể thực hiện.
4. chọn option "Copy"



Dán thư mục/file từ clipboard

Qui trình dán thư mục/file từ clipboard vào thư mục chứa như sau :

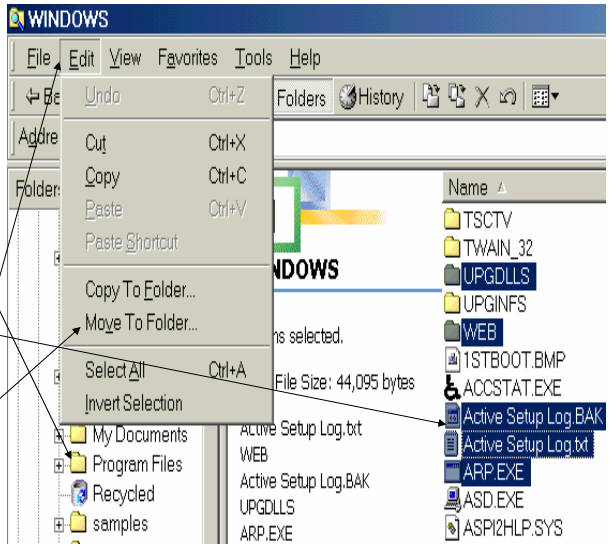
1. duyệt cây thư mục trong cửa sổ bên trái và chọn thư mục mà ở đó bạn muốn dán thư mục/file.
2. chọn menu Edit để hiển thị menu các tác vụ có thể thực hiện.
3. chọn option "Paste"



Di chuyển (move) thư mục/file

Việc di chuyển thư mục/file được thực hiện bởi 3 tác vụ copy/paste/delete như đã được trình bày. Qui trình move thư mục/file khác như sau :

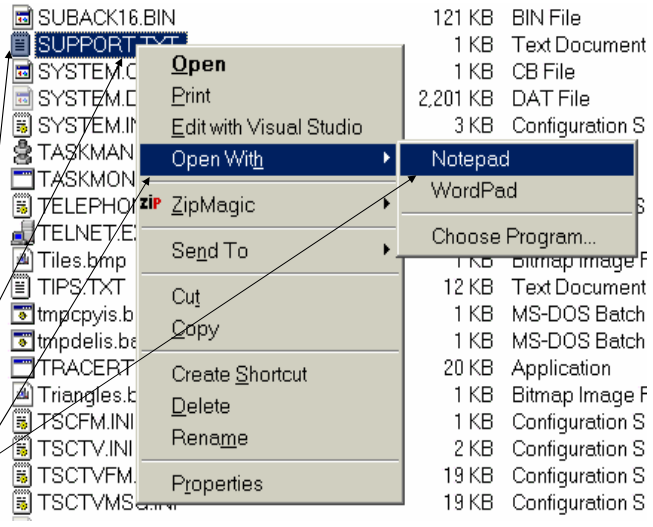
1. duyệt cây thư mục trong cửa sổ bên trái và chọn thư mục mà ở đó bạn muốn copy thư mục/file.
2. chọn các phần tử cần copy trong ListCtrl bên phải.
3. chọn menu Edit để hiển thị menu các tác vụ có thể thực hiện.
4. chọn option "Move to Folder" và xác định thư mục đích.



Load file vào bộ nhớ để hiệu chỉnh

Qui trình chạy ứng dụng và load file vào bộ nhớ để hiệu chỉnh nội dung file như sau :

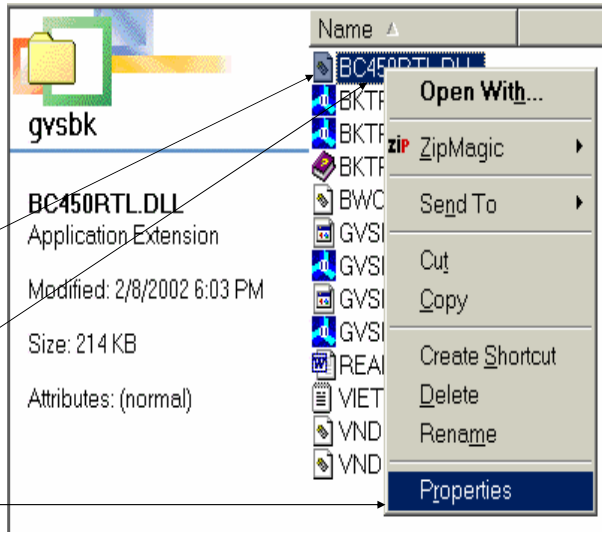
1. duyệt cây thư mục trong cửa sổ bên trái và chọn thư mục chứa file cần hiệu chỉnh.
2. chọn file cần hiệu chỉnh trong ListCtrl bên phải.
3. ấn phải chuột vào file chọn để hiển thị menu các tác vụ có thể thực hiện.
4. chọn option "Open with" và xác định ứng dụng được dùng để hiệu chỉnh file.



Hiển thị cửa sổ thông tin về file/thư mục

Qui trình làm hiển thị cửa sổ thông tin thư mục/file như sau :

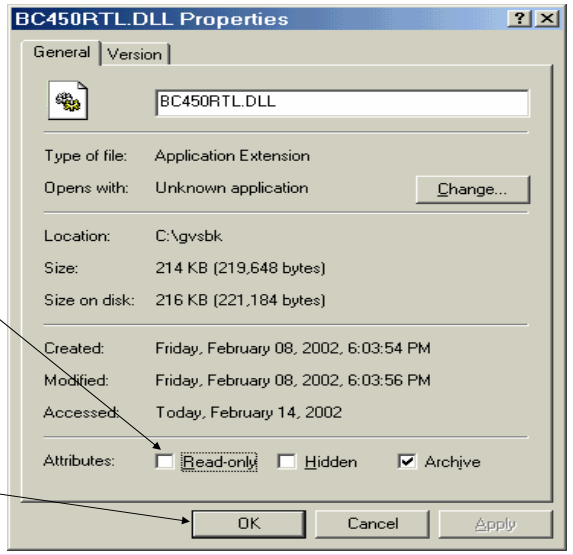
1. duyệt cây thư mục trong cửa sổ bên trái và chọn thư mục/file cần hiển thị thông tin.
2. chọn thư mục/file cần hiển thị thông tin trong ListCtrl bên phải.
3. ấn phải chuột vào file chọn để hiển thị menu các tác vụ có thể thực hiện.
4. chọn option "Properties" để làm hiển thị cửa sổ thông tin của thư mục/file tương ứng.



Xem và hiệu chỉnh thuộc tính file/thư mục

Khi cửa sổ thông tin của thư mục/file đã được hiển thị, chọn trang general/Version để thấy các thông tin tương ứng. Trang bên phải là trang General.

1. xem các thuộc tính file.
2. nếu muốn thay đổi thuộc tính nào đó, ấn chuột vào checkbox tương ứng. Thuộc tính sẽ chuyển từ không thành có hay ngược lại.
3. nếu muốn cập nhật các hiệu chỉnh thì ấn chuột vào button OK.



MÔN TIN HỌC

Chương 3

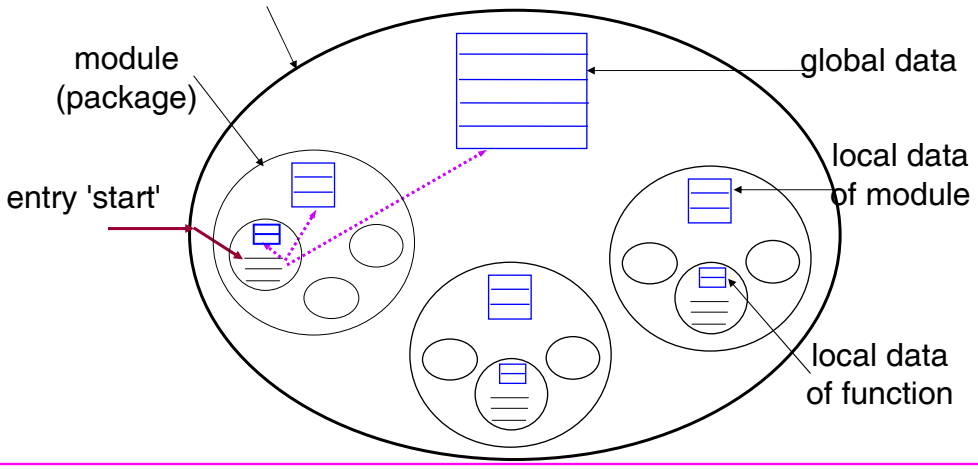
TỔNG QUÁT VỀ LẬP TRÌNH BẰNG VISUAL BASIC

3.1 Các khái niệm cơ bản về mô hình hướng đối tượng
3.2 Các đối tượng giao diện của VB 6.0
3.3 Hiệu chỉnh thuộc tính của các đối tượng giao diện
3.4 Tạo thủ tục xử lý sự kiện của các đối tượng giao diện.

3.1 Các khái niệm cơ bản về mô hình hướng đối tượng

Hình vẽ sau đây tổng kết cấu trúc của 1 ứng dụng được lập trình cấu trúc :

Chương trình = cấu trúc dữ liệu + giải thuật



Từ lập trình cấu trúc đến OOP

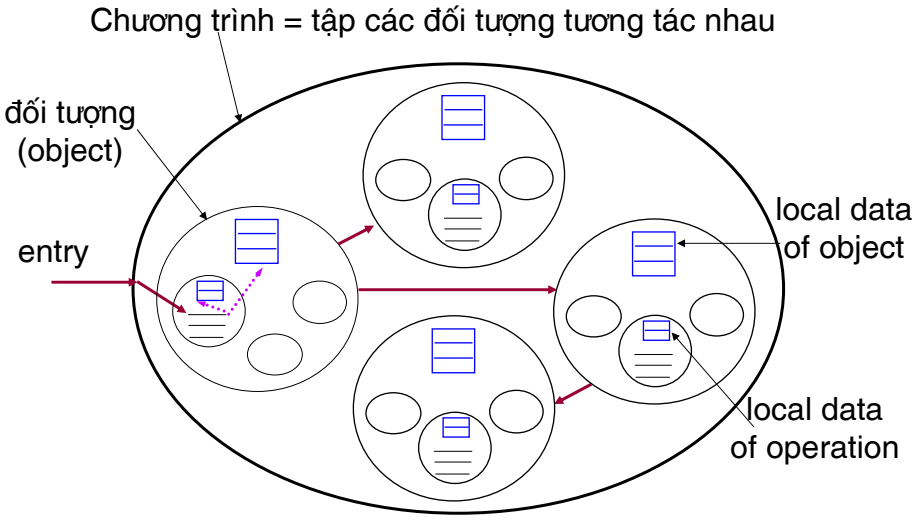
Xét cấu trúc chương trình cổ điển của slide trước, ta thấy có 2 nhược điểm chính sau :

1. rất khó đảm bảo tính nhất quán và đúng đắn của dữ liệu toàn cục vì bất kỳ lệnh nào trong hàm nào cũng có thể truy xuất chúng.
2. nếu chương trình cần đồng thời nhiều 'instance' của cùng 1 module thì cơ chế lập trình cấu trúc không cho phép tạo tự động các 'instance' này.

Để khắc phục 2 nhược điểm chính trên (và bổ sung nhiều ưu điểm khác), ta sẽ lập trình theo hướng đối tượng (OOP - Object Oriented Programming) trong đó chương trình là 1 tập các đối tượng sống tương tác nhau (xem slide kế tiếp).

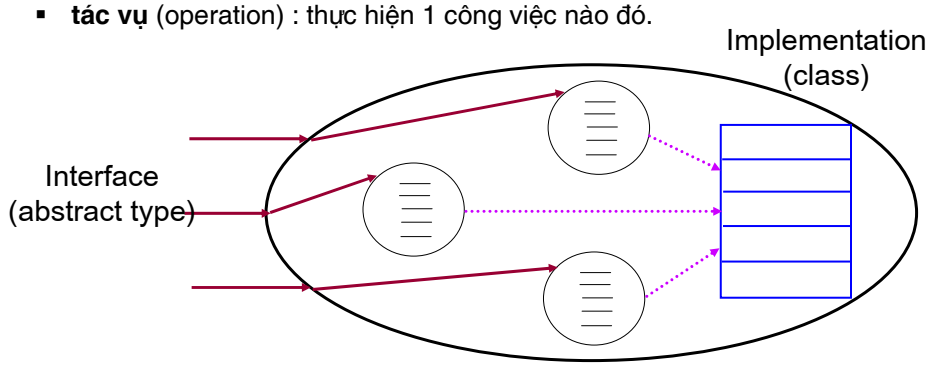
Visual Basic là ngôn ngữ hỗ trợ việc lập trình theo hướng đối tượng, hơn nữa VB còn là môi trường lập trình trực quan (visual) nên rất dễ dùng.

Cấu trúc của 1 ứng dụng OOP



Đối tượng (Object)

- Mô hình đối tượng quan niệm chương trình bao gồm các đối tượng sinh sống và tương tác với nhau.
- Đối tượng bao gồm nhiều thành phần, mỗi thành phần thuộc 1 trong 2 loại :
  - **thuộc tính** (attribute) : mang 1 giá trị nhất định tại từng thời điểm.
  - **tác vụ** (operation) : thực hiện 1 công việc nào đó.



Kiểu trừu tượng (Abstract type)

- Abstract type (*type*) định nghĩa interface sử dụng đối tượng. Ta dùng tên nhận dạng để đặt tên cho kiểu và để nhận dạng nó.
- Interface là tập hợp các 'entry' mà bên ngoài có thể giao tiếp với đối tượng.
- Ta dùng signature để định nghĩa mỗi 'entry'. Signature gồm :
  - tên tác vụ (operation, function)
  - danh sách tham số hình thức, mỗi tham số được đặc tả bởi 3 thuộc tính : tên, type và chiều di chuyển (IN, OUT, INOUT).
  - đặc tả chức năng của tác vụ (thường ở dạng chú thích).
- Ta dùng tên của abstract type (chứ không phải class) để đặc tả kiểu cho biến, thuộc tính, tham số hình thức.
- User không cần quan tâm đến class (hiện thực cụ thể) của đối tượng.

Class (Implementation)

- Ta dùng tên nhận dạng để đặt tên cho class và để nhận dạng nó. Class định nghĩa chi tiết hiện thực đối tượng :
  - định nghĩa các thuộc tính dữ liệu, mỗi thuộc tính được đặc tả bởi các thông tin về nó như tên nhận dạng, kiểu dữ liệu, tầm vực truy xuất,... Kiểu của thuộc tính có thể là type cổ điển (số nguyên, thực, ký tự, chuỗi ký tự,...) hay 'abstract type', trong trường hợp sau thuộc tính sẽ là tham khảo đến đối tượng khác. Trạng thái của đối tượng là tập giá trị tại thời điểm tương ứng của tất cả thuộc tính của đối tượng. Trong thời gian tồn tại và hoạt động, trạng tái của đối tượng sẽ thay đổi.
  - 'coding' các tác vụ (miêu tả giải thuật chi tiết về hoạt động của tác vụ) và các 'internal function'.
- Định nghĩa các tác vụ tạo (create) và xóa (delete) đối tượng.
- Định nghĩa các tác vụ 'constructor' và 'destructor'.
- User không cần quan tâm đến class của đối tượng.

Tính bao đóng (encapsulation)

- Bao đóng : che dấu mọi chi tiết hiện thực của đối tượng, không cho bên ngoài thấy và truy xuất $\Rightarrow$ đảm bảo tính độc lập cao giữa các đối tượng, nghĩa là độ phụ thuộc (hay tính ghép nối - coupling giữa các đối tượng) rất thấp, nhờ đó dễ bảo trì, phát triển ứng dụng :
  - che dấu các thuộc tính dữ liệu : nếu cần cho phép truy xuất 1 thuộc tính nào đó từ bên ngoài, ta tạo 2 tác vụ get/set tương ứng để giám sát việc truy xuất và che dấu chi tiết hiện thực bên trong.
  - che dấu chi tiết hiện thực các tác vụ.
  - che dấu các 'internal function' và sự hiện thực của chúng.

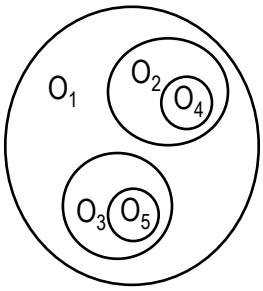
Tính thừa kế (inheritance)

- Viết 1 ứng dụng OOP là định nghĩa các type/class của các đối tượng cấu thành ứng dụng.
- Tính thừa kế cho phép giảm nhẹ công sức định nghĩa type/class : ta có thể định nghĩa các type/class không phải từ đầu mà bằng cách kế thừa các type/class có sẵn, ta chỉ định nghĩa thêm các chi tiết mới mà thôi (thường khá ít).
  - Đa thừa kế hay đơn thừa kế.
  - Mối quan hệ supertype/subtype và superclass/subclass.
  - có thể 'override' sự hiện thực các tác vụ của class cha, kết quả override chỉ có tác dụng trên các đối tượng của class con.
  - Đối tượng của class con có thể đóng vai trò của đối tượng cha nhưng ngược lại thì không đúng.

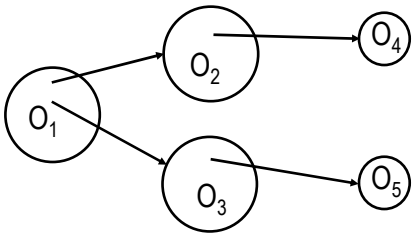
Tính bao gộp (aggregation)

- 1 đối tượng có thể chứa nhiều đối tượng khác nhờ mối quan hệ bao gộp 1 cách đệ qui giữa các đối tượng.
- Có 2 góc nhìn về tính bao gộp : ngữ nghĩa và hiện thực.

Góc nhìn ngữ nghĩa



Góc nhìn hiện thực



Thông điệp (Message)

- Thông điệp là 1 phép gọi tác vụ của 1 đối tượng từ 1 tham khảo.
- Thông điệp bao gồm 3 phần :
  - tham khảo đến đối tượng đích.
  - tên tác vụ muốn gọi.
  - danh sách tham số thực cần truyền theo (hay nhận về từ) tác vụ.
  - ví dụ : aCircle.Draw (pWnd)
  - truy xuất thuộc tính trong interface :
    aCircle.Radius = 10 ≡ aCircle.SetRadius(10)
    r = aCircle.Radius ≡ r = aCircle.GetRadius()
- Thông điệp là phương tiện giao tiếp (hay tương tác) duy nhất giữa các đối tượng.

Hai thành phần chính của 1 ứng dụng

Xem lại slide 15 miêu tả qui trình tổng quát của việc dùng máy tính giải quyết 1 vấn đề ngoài đời, ta thấy 1 ứng dụng gồm 2 phần thành phần chính :

1. **giao diện người dùng** : là phương tiện cho người dùng tương tác với chương trình để nhập/xuất dữ liệu, để điều khiển/giám sát hoạt động của chương trình. Trong OOP, giao diện người dùng là tập các đối tượng giao diện như form, mỗi form chứa nhiều đối tượng nhỏ hơn như menu, toolbar, button, textedit, listbox, treeview...
2. **giải thuật xử lý bên trong** : được thể hiện bởi các method của các đối tượng giao diện và các đối tượng bên trong ứng dụng. Mỗi method là danh sách các lệnh thực thi (cấu trúc điều khiển) để miêu tả giải thuật mà tác vụ tương ứng thực hiện.

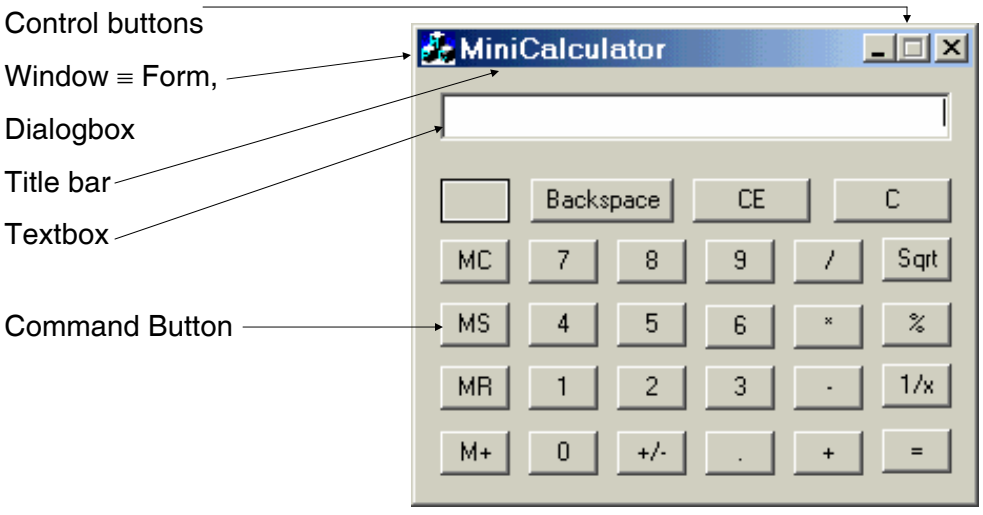
Thiết kế trực quan các đối tượng giao diện

Định nghĩa các đối tượng giao diện bằng cách viết code tường minh là 1 công việc rất khó khăn và tốn nhiều công sức, thời gian.

Để giảm nhẹ công sức định nghĩa các đối tượng giao diện, các môi trường lập trình trực quan (visual) đã viết sẵn 1 số đối tượng giao diện thường dùng và cung cấp công cụ để người lập trình thiết kế trực quan giao diện của ứng dụng bằng cách tích hợp các đối tượng giao diện có sẵn này : người lập trình đóng vai trò họa sĩ để vẽ/hiệu chỉnh kích thước, di chuyển vị trí các phần tử giao diện cần cho ứng dụng.

Ngoài ra môi trường trực quan còn cho phép người lập trình tự tạo các đối tượng giao diện mới (ActiveX Control) để dùng trong các ứng dụng được viết sau đó. Qui trình viết ứng dụng theo cơ chế này được gọi là viết ứng dụng bằng cách lắp ghép các linh kiện phần mềm, nó giống như việc lắp máy tính từ các linh kiện phần cứng như CPU, RAM, disk, keyboard, monitor,...⇒ rất dễ dàng và nhanh chóng.

3.2 Các đối tượng giao diện có trong VB



Các đối tượng giao diện có trong VB (tt)

Label

DriveListBox

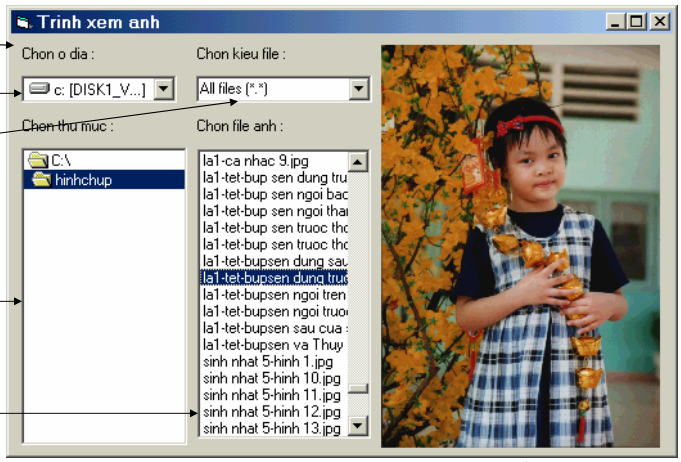
Combobox ≡

Textbox + ListBox

DirListBox

FileListBox ≅ ListBox

Image ≅ Picture

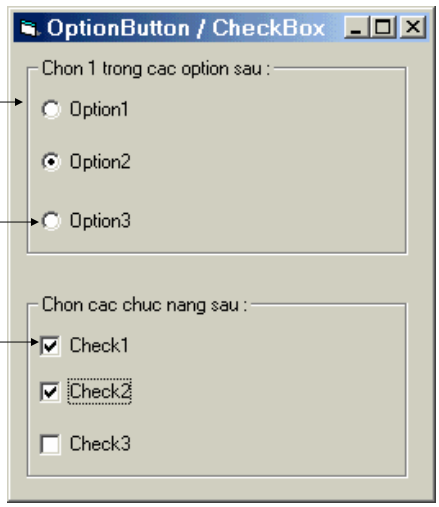


Các đối tượng giao diện có trong VB (tt)

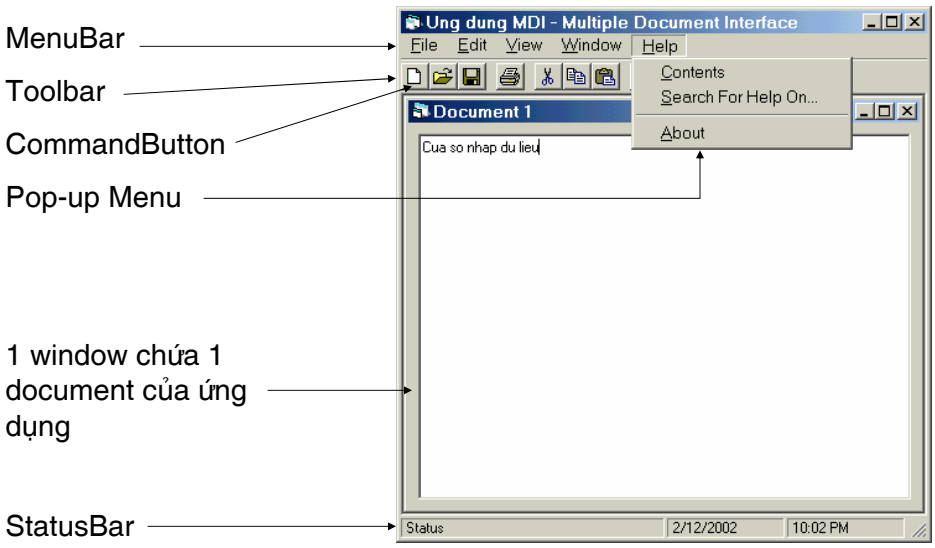
Frame

OptionButton

Checkbox



Các đối tượng giao diện có trong VB (tt)



Các tính chất chung của các đối tượng giao diện

Đối tượng giao diện có những tính chất giống như đối tượng bình thường, ngoài ra chúng còn có 1 số đặc điểm riêng.

Đối tượng giao diện cũng được cấu thành từ 2 loại thành phần : thuộc tính và tác vụ.

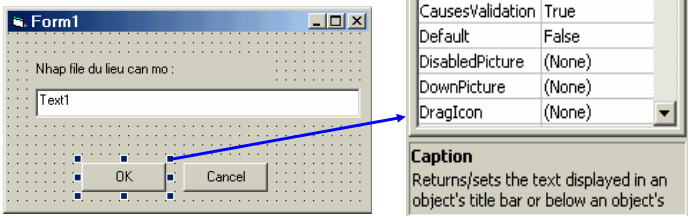
Mỗi đối tượng giao diện chứa khá nhiều thuộc tính liên quan đến nhiều loại trạng thái khác nhau :

- thuộc tính 'Name' : đây là thuộc tính đặc biệt, xác định tên nhận dạng của đối tượng, giá trị của thuộc tính này sẽ trở thành biến tham khảo đến đối tượng, code của ứng dụng sẽ dùng biến này để truy xuất đối tượng.
- các thuộc tính xác định vị trí và kích thước : Left, Top, Height, Width...
- các thuộc tính xác định tính chất hiển thị : Caption, Picture, BackColor,...
- các thuộc tính xác định hành vi : Enable, ...
- ...

3.3 Hiệu chỉnh thuộc tính của các đối tượng giao diện

Khi tạo trực quan 1 đối tượng giao diện, môi trường đã gán giá trị ban đầu cho các thuộc tính, thường ta chỉ cần thay đổi 1 vài thuộc tính là đáp ứng được yêu cầu riêng. Có 2 cách để hiệu chỉnh giá trị 1 thuộc tính :

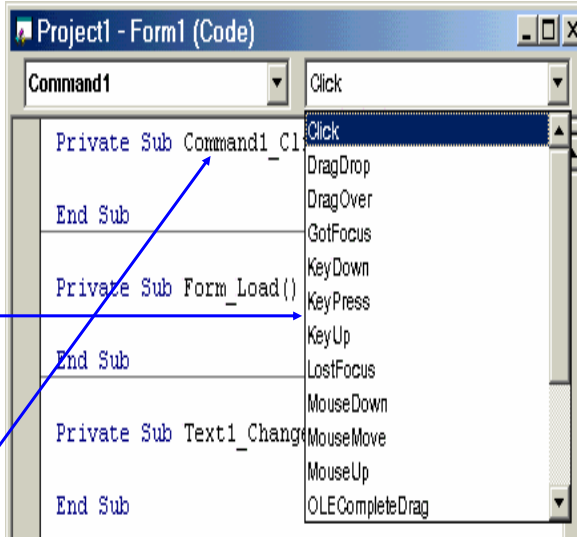
1. trực quan thông qua cửa sổ thuộc tính của đối tượng giao diện.
2. lập trình truy xuất thuộc tính của đối tượng giao diện.



3.4 Sự kiện - Thủ tục xử lý sự kiện

Mỗi đối tượng giao diện có khá nhiều tác vụ (operation), hầu hết chúng được gọi là thủ tục xử lý sự kiện vì cơ chế gọi thủ tục này chủ yếu là trực tiếp từ người dùng ứng dụng thông qua sự tương tác trực tiếp với đối tượng, từ đó tạo sự kiện kích khởi thủ tục xử lý tương ứng chạy.

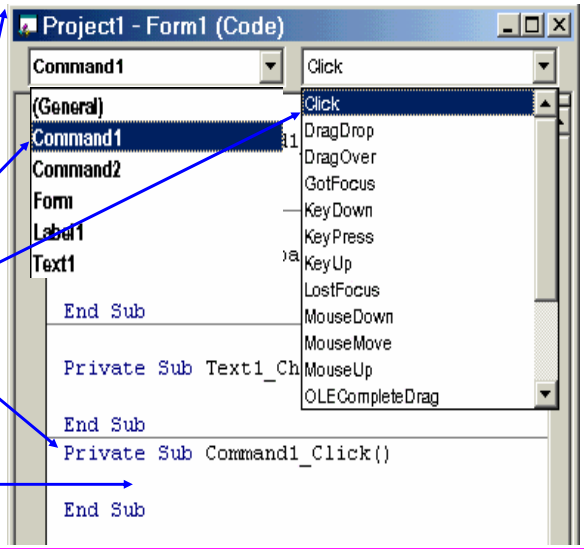
Thí dụ khi ta ấn chuột vào button tên "Command1", hệ thống tạo ra sự kiện "Click" để kích khởi thủ tục Command1_Click() chạy.



Cách tạo hàm xử lý sự kiện của đối tượng

Qui trình tổng quát của việc tạo thủ tục xử lý cho 1 sự kiện nào đó của 1 đối tượng :

1. chọn menu View.Code để hiển thị cửa sổ code.
2. chọn tên đối tượng liên quan trong danh sách các đối tượng.
3. chọn sự kiện cần tạo thủ tục xử lý trong danh sách các sự kiện, template của thủ tục xử lý sẽ được tạo tự động.
4. sử dụng kiến thức về giải thuật & cú pháp ngôn ngữ VB để viết code cho thủ tục xử lý.



Tổng kết qui trình viết 1 ứng dụng bằng VB

1. Trước hết phải nắm bắt yêu cầu phần mềm để xác định các chức năng mà ứng dụng phải cung cấp cho người dùng.
2. Phân tích sơ lược từng chức năng và tìm ra các class phân tích cấu thành chức năng tương ứng.
3. Thiết kế chi tiết các class phân tích : xác định các thuộc tính và các tác vụ cũng như phác họa giải thuật của từng tác vụ.
4. Hiện thực phần mềm bằng VB gồm 2 công việc chính :
   1. **thiết kế trực quan các form giao diện người dùng** : mỗi form chứa nhiều phần tử giao diện, các phần tử giao diện thường đã có sẵn, nếu không ta phải tạo thêm 1 số đối tượng giao diện mới (ActiveX Control). Ứng với mỗi phần tử giao diện vừa tạo ra, nên thiết lập giá trị đầu cho thuộc tính "Name" và 1 vài thuộc tính cần thiết.
   2. **tạo thủ tục xử lý sự kiện** cho các sự kiện cần thiết trên các phần tử giao diện rồi viết code cho từng thủ tục xử lý sự kiện vừa tạo ra.

MÔN TIN HỌC

Chương 4

QUI TRÌNH THIẾT KẾ TRỰC QUAN GIAO DIỆN CỦA ỨNG DỤNG

4.1 Dự Án Và Ứng Dụng
4.2 Tạo/xóa đối tượng giao diện.
4.3 Hiệu chỉnh giá trị thuộc tính của đối tượng giao diện
4.4 Tạo menubar
4.5 Tạo Toolbar
4.6 Tạo và viết thủ tục xử lý sự kiện

4.1 Dự Án Và Ứng Dụng

1 ứng dụng VB được cấu thành từ nhiều đối tượng thuộc nhiều loại :

- Các phần tử giao diện
- Các "class module", mỗi class đặc tả 1 loại đối tượng cần dùng cho giải thuật của chương trình.
- Các đối tượng khác như các thư viện liên kết động, các database,...

Để quản lý ứng dụng được dễ dàng ta sử dụng phương tiện "Dự án" (Project). Dự án là 1 cây thứ bậc các phần tử cấu thành ứng dụng. Viết ứng dụng là qui trình tạo dự án, thêm/bớt, hiệu chỉnh từng phần tử trong dự án.

Thao tác để thực hiện các tác vụ trên khá giống với các thao tác mà ta đã biết trên hệ thống file thứ bậc của máy tính.

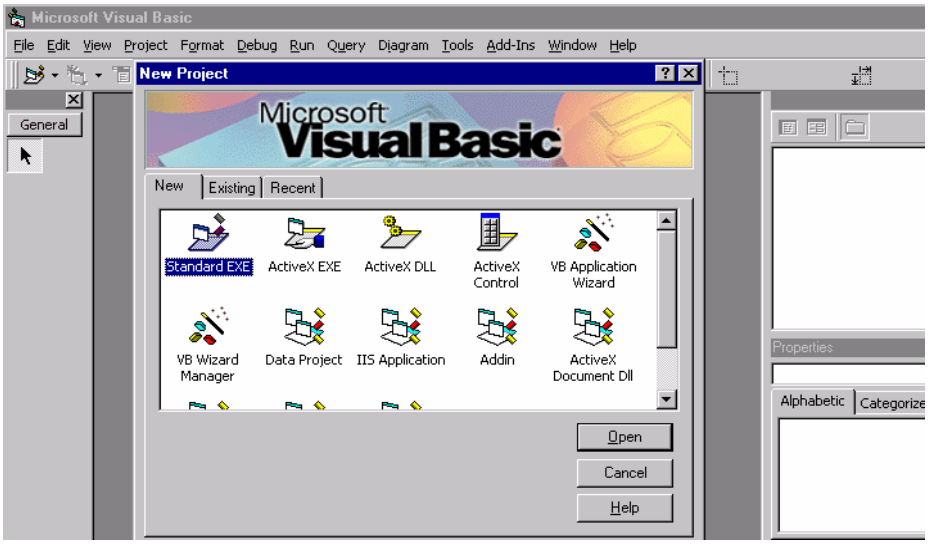
Khởi động VB 6.0

VB là 1 ứng dụng như bao ứng dụng khác. Để khởi động 1 ứng dụng, ta có nhiều cách khau nhau :

- chọn menu **Start.Programs.Microsoft Visual Basic 6.0.Microsoft Visual Basic 6.0.**
- Ấn kép chuột vào icon shortcut của VB trên màn hình desktop (ta phải tạo trước icon shortcut này).
- chọn menu Start.Run, rồi nhập hàng lệnh chạy ứng dụng, thí dụ như "c:\Program Files\Microsoft Visual Studio\VB98\VB6.exe".
- dùng trình quản lý hệ thống file WE, duyệt đến thư mục chứa file chương trình VB (thí dụ c:\Program Files\Microsoft Visual Studio\VB98), ấn kép vào file chương trình VB6.exe.

Sau khi VB được khởi động, ta thường thấy cửa sổ màn hình như sau:

Cửa sổ khởi động VB



Tạo mới dự án

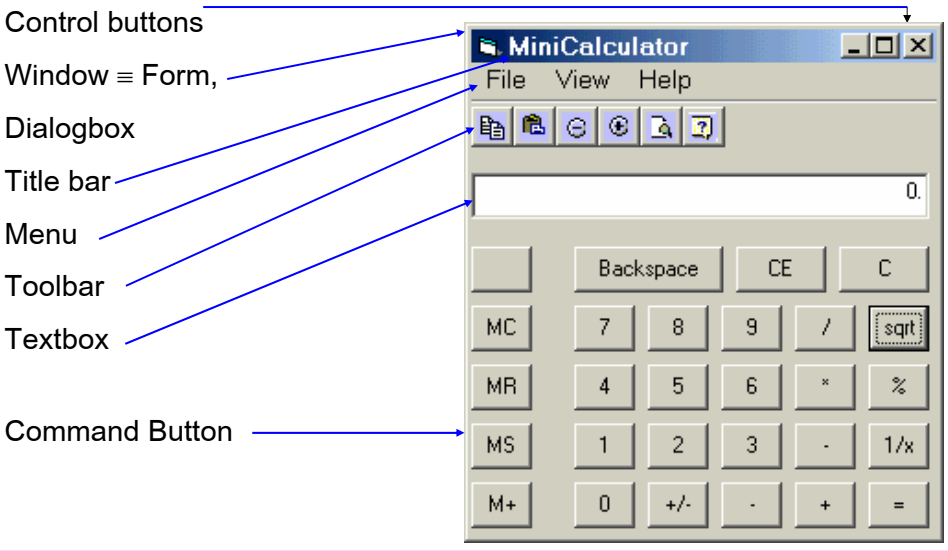
Cửa sổ New Project của VB có ba thẻ (Tab) :

- **New** : tạo mới một dự án (tab này được chọn default)
- **Existing** : Mở 1 dự án đã có sẵn trên máy (dự án cũ nào đó).
- **Recent** : Mở 1 dự án trong n dự án gần hiện tại nhất.

Với tab New được chọn, bạn có thể tạo 1 dự án theo 1 loại nào đó, nhưng đối với các ứng dụng thông thường ta sẽ dùng loại dự án "Standard EXE". Ấn kép vào icon "Standard EXE" để tạo mới dự án tương ứng. 1 form mới được tạo ra tự động để bạn có thể thiết kế trực quan form giao diện này.

Qui trình thiết kế giao diện là tuần tự thiết kế từng form theo yêu cầu, nếu muốn tạo mới 1 form khác (hay 1 đối tượng nào đó vào dự án), bạn ấn kép chuột vào cửa sổ Project, dời chuột đến menu "Add", rồi chọn mục "Form" trong danh sách.

Thí dụ về form thiết kế : MiniCalculator



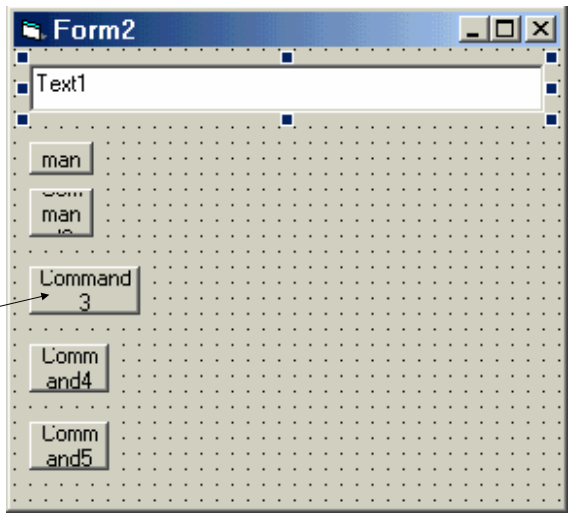
4.2 Tạo 1 đối tượng giao diện trên form

- Để hiển thị và làm việc trực quan với 1 form, ấn kép chuột vào mục tên form trong cửa sổ Project.
- Để tạo mới 1 đối tượng giao diện trong form, dùng chuột chọn icon tương ứng với đối tượng trong cửa sổ Toolbox rồi vẽ đối tượng ở vị trí và kích thước mong muốn trên form.
- Bạn cũng có thể tạo mới đối tượng giao diện dùng cơ chế sinh sản vô tính : chọn đối tượng đã có, ấn button Copy trên Toolbar rồi ấn button Past trên Toolbar, đối tượng mới sinh ra giống y như đối tượng có sẵn (nên đặt lại tên khác bằng cách chọn button "No" trong hộp thoại yêu cầu sau khi ấn icon Past). Đây là 1 trong nhiều cách để tạo nhiều đối tượng có kích thước giống hệt nhau.

Thí dụ slide sau miêu tả trạng thái của form sau khi ta vẽ được 1 textbox hiển thị số và 5 button bên trái nhất của máy tính.

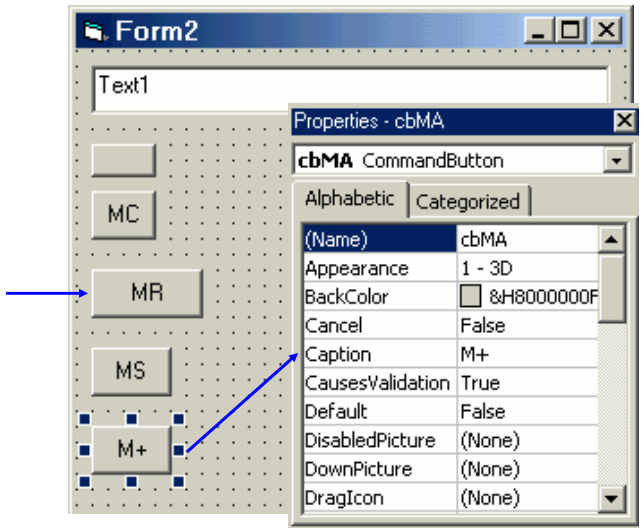
4.3 Thiết lập giá trị cho các thuộc tính

- Để dễ cân chỉnh vị trí và kích thước của các đối tượng, ta nên thiết lập các thuộc tính cơ bản sau : "Name", "Caption". Thuộc tính "Name" được dùng để truy xuất đối tượng lúc lập trình, còn thuộc tính "Caption" được hiển thị trên phần tử (không phải đối tượng nào cũng có Caption).



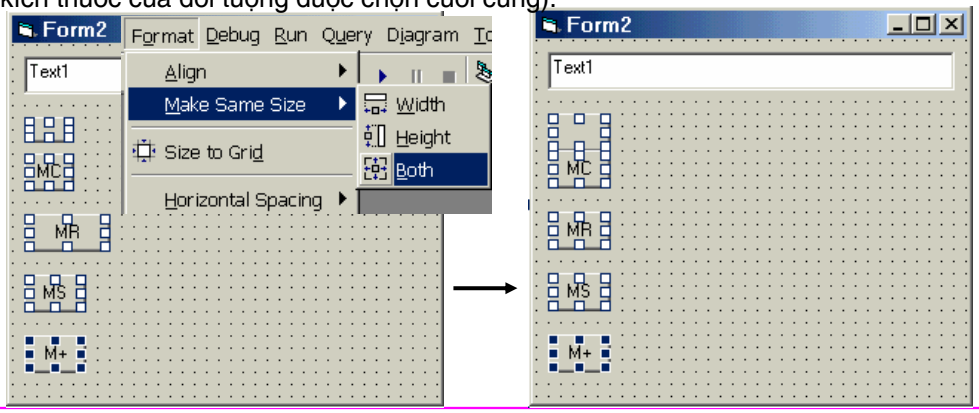
Thiết lập giá trị cho các thuộc tính (tt)

- Để xem và hiệu chỉnh giá trị của các thuộc tính của 1 đối tượng giao diện, bạn ấn chuột chọn đối tượng, cửa sổ Properties bên phải màn hình sẽ hiển thị các thuộc tính của đối tượng được chọn. Bạn duyệt các thuộc tính từ trên xuống và nhập vào giá trị mới cho thuộc tính mong muốn.



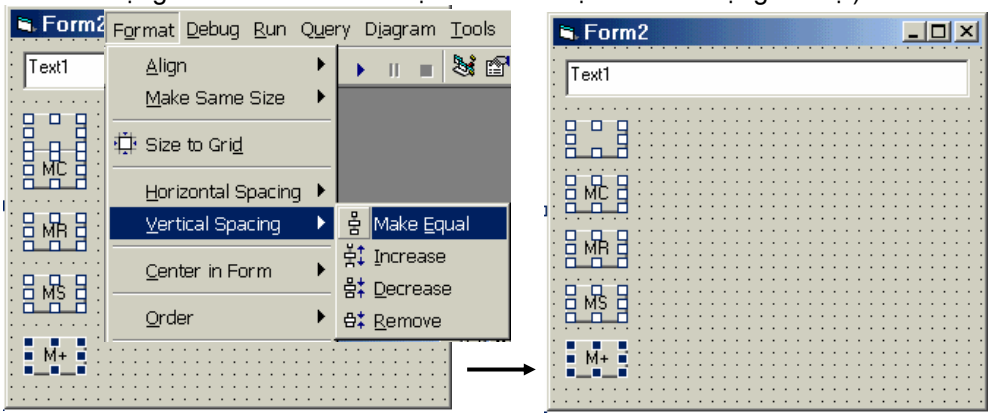
Cân chỉnh kích thước các đối tượng

Nếu vẽ bằng tay tuần tự các đối tượng thì khó lòng đảm bảo kích thước của chúng bằng nhau, do đó bạn nên dùng cơ chế sinh sản vô tính (Copy-Paste). Tuy nhiên nếu lở tạo bằng tay các đối tượng rồi thì để làm kích thước nhiều đối tượng giống y nhau, bạn chọn các đối tượng rồi chọn menu Format.Make Same Size.Both (bằng kích thước của đối tượng được chọn cuối cùng).



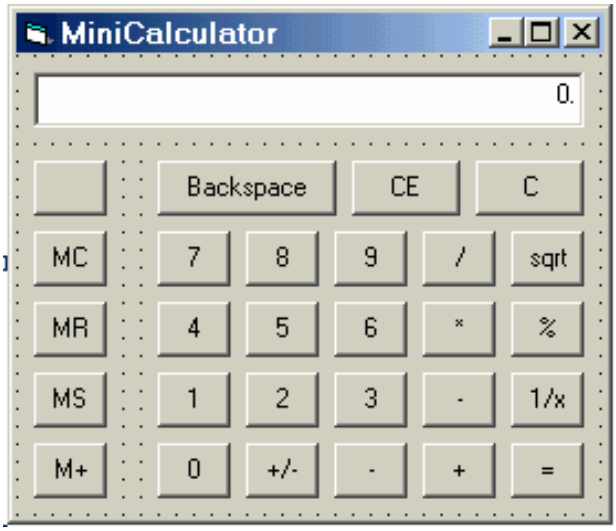
Đồng chỉnh vị trí các đối tượng

Tương tự, nếu vẽ bằng tay tuần tự các đối tượng thì khó lòng đảm bảo khoảng cách giữa chúng đều nhau. Để khoảng cách dọc giữa các đối tượng đều nhau, bạn chọn các đối tượng rồi chọn menu Format.Vertical Spacing.Make Equal (cố định vị trí 2 đối tượng xa nhất theo chiều dọc rồi chỉnh dọc các đối tượng còn lại).



Kết quả tạm thời của form MiniCalculator

Với qui trình tạo đối tượng, thiết lập các thuộc tính cần thiết và chỉnh dạng các đối tượng giao diện như đã được trình bày, bạn tiếp tục tạo các đối tượng còn lại của form MiniCalculator. Kết quả như sau :



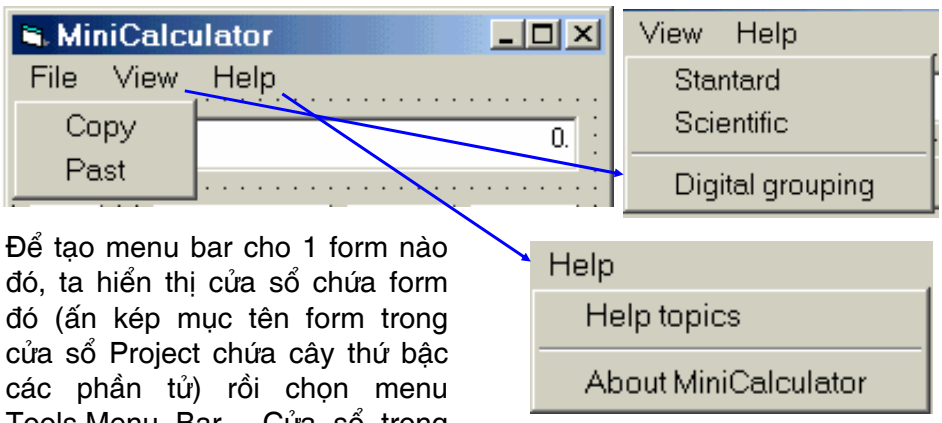
Danh sách thuộc tính các đối tượng

Danh sách giá trị các thuộc tính được thiết lập cho các đối tượng (sẽ được tham khảo bởi code chương trình được viết sau) :

- Caption = , Name = cmdMemStatus
- Caption = MC, Name = cmdMC
- Caption = MR, Name = cmdMR
- Caption = MS, Name = cmdMS
- Caption = MA, Name = cmdMA
- Caption = 0, Name = cmd0
- Caption = 1, Name = cmd1
- Caption = 2, Name = cmd2
- Caption = 3, Name = cmd3
- Caption = 4, Name = cmd4
- Caption = 5, Name = cmd5
- Caption = 6, Name = cmd6
- Caption = 7, Name = cmd7
- Caption = 8, Name = cmd8
- Caption = 9, Name = cmd9
- Caption = +, Name = cmdAdd
- Caption = -, Name = cmdSub
- Caption = *, Name = cmdMul
- Caption = /, Name = cmdDiv
- Caption = +/-, Name = cmdPosNeg
- Caption = ., Name = cmdPoint
- Caption = =, Name = cmdEqual
- Caption = 1/x, Name = cmd1x
- Caption = %, Name = cmdPercent
- Caption = sqrt, Name = cmdSqrt
- Caption = C, Name = cmdC
- Caption = CE, Name = cmdCE
- Caption = Backspace, Name = cmdBack
- Text = 0., Name = txtDisplay

4.4 Thiết kế menu bar cho form giao diện

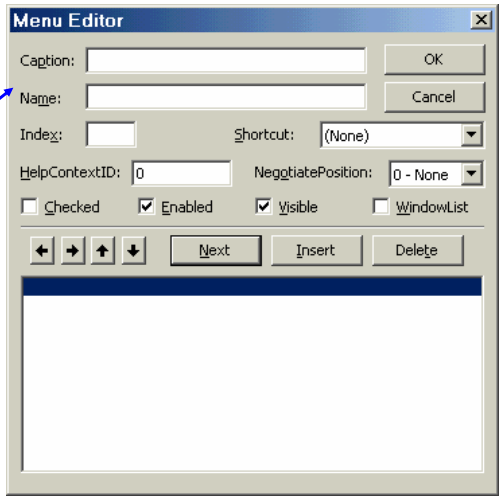
Giả sử form MiniCalculator cần có 1 hệ thống menu như sau :



Để tạo menu bar cho 1 form nào đó, ta hiển thị cửa sổ chứa form đó (ấn kép mục tên form trong cửa sổ Project chứa cây thứ bậc các phần tử) rồi chọn menu Tools.Menu Bar... Cửa sổ trong slide sau sẽ hiện lên :

Dùng Menu Editor để thiết kế menu bar

- VB luôn tạo sẵn 1 mục mới trống ở hàng cuối của danh sách. Thêm 1 phần tử mới là chọn mục mới này và nhập ít nhất 2 thuộc tính Caption và Name của nó.
- Button Next cho phép dời mục chọn xuống 1 hàng.
- Button Insert cho phép chèn 1 mục trống vào trước mục được chọn hiện hành.
- Button Delete cho phép xóa mục được chọn.
- Các button ↑,↓ cho phép dời mục được chọn đi lên hay xuống 1 vị trí.
- Các button →,← cho phép dời mục được chọn vô thêm hay ra bớt 1 cấp trong hệ thống cây phân cấp menu.



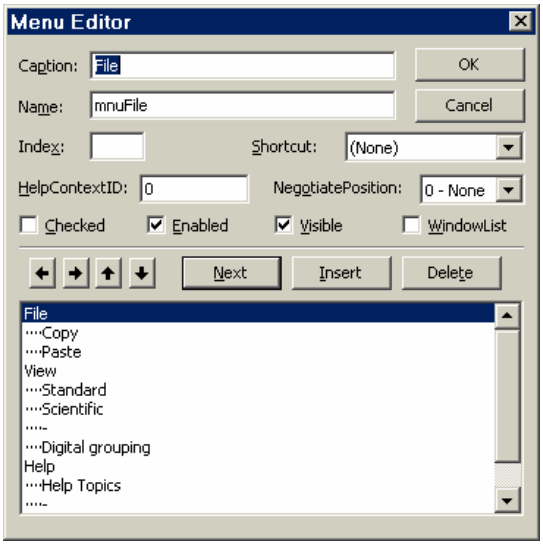
Dùng Menu Editor để thiết kế menu bar (tt)

Dựa vào đặc tả menu bar của slide trước đây, nhập lần lượt các mục sau :

- Caption = File, Name = mnuFile
  - Caption = Copy, Name = mnuFileCopy, ấn button → để vô thêm 1 cấp
  - Caption = Paste, Name = mnuFilePaste
- Caption = View, Name = mnuView, ấn button ← để ra 1 cấp
  - Caption = Standard, Name = mnuViewStand, ấn button → để vô thêm 1 cấp
  - Caption = Scientific, Name = mnuViewScien
  - Caption = -, Name = mnuViewBar
  - Caption = Digital grouping, Name = mnuViewDigital
- Caption = Help, Name = mnuHelp, ấn button ← để ra 1 cấp
  - Caption = Help Topics, Name = mnuHelpTopics, ấn button → để vô thêm 1 cấp
  - Caption = -, Name = mnuHelpBar
  - Caption = About MiniCalculator, Name = mnuHelpAbout.

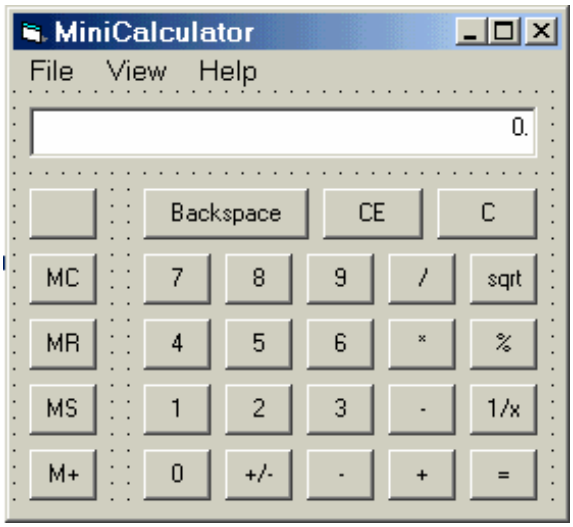
Dùng Menu Editor để thiết kế menu bar (tt)

Sau khi đặc tả xong menu, cửa sổ menu editor có dạng như sau. Lưu ý lúc này bạn vẫn chưa thấy menu 1 cách trực quan :



Kết quả của hoạt động thiết kế menubar

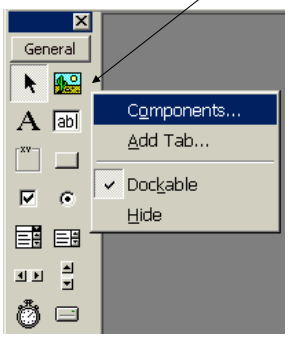
Sau khi tạo menu xong, hãy ấn nút OK để đóng tiện ích "Menu Editor", form giao diện của chương trình sẽ giống như hình bên :



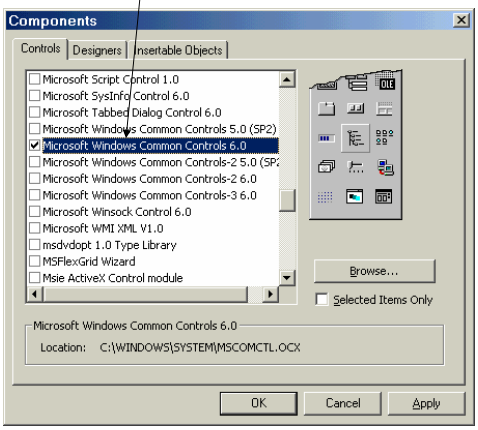
4.5 Thiết kế Toolbar cho form giao diện

Để tạo Toolbar cho 1 form trong project, trước hết ta phải thêm tập các điều khiển "Window Common Controls 6.0" vào cửa sổ Toolbox của project :

1. ấn phải chuột vào vị trí trống của Toolbox, chọn mục Components



2. chọn tab Controls, duyệt và chọn mục tương ứng, chọn OK.



3. các icon mới được thêm vào Tool-box

Qui trình tạo Toolbar của form

- Toolbar là 1 cửa sổ chứa nhiều button (icon), mỗi button cho phép thực hiện 1 chức năng của ứng dụng. Các button có kích thước đều nhau, nên kết hợp 1 ảnh bitmap với từng button, nội dung ảnh làm sao gợi ý cho người dùng về chức năng tương ứng (thí dụ ảnh dạng cái kéo gợi ý chức năng Cut,...).

1. Công việc đầu tiên cần thực hiện là dùng 1 trình soạn thảo đồ họa (Paint, CorelDraw,...) để thiết kế (vẽ) từng ảnh bitmap gợi ý cho từng button trong Toolbar. Bạn có thể dùng trình "Screen Capture" cắt các icon có sẵn của ứng dụng đang chạy và dán vào vùng soạn thảo ảnh của trình soạn thảo đồ họa. Sau khi soạn xong 1 ảnh, ta cất ảnh lên file dạng *.bmp. Lưu ý rằng các ảnh phải có cùng kích thước (thí dụ 16*16, 20*20,32*32,...). Slide kế miêu tả cửa sổ của trình soạn thảo đồ họa Paint.

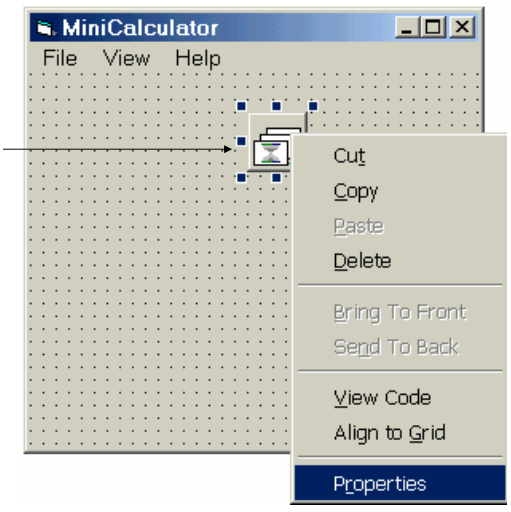
Vẽ ảnh cho button bằng trình Paint

- soạn thảo 1 button là vẽ từng pixel ảnh. Để dễ vẽ, bạn nên phóng to ảnh lên khoảng 400% trở lên.
- trước khi vẽ 1 pixel, hãy chọn màu vẽ thích hợp.
- sau khi vẽ xong, dùng menu File.Save As để cất ảnh lên file thích hợp :
  - copy.bmp
  - paste.bmp
  - standard.bmp
  - scientific.bmp
  - help.bmp
  - about.bmp

Qui trình tạo Toolbar của form (tt)

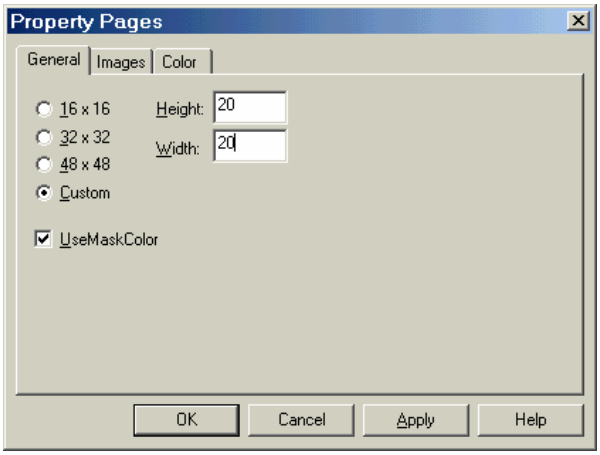
2. Dùng qui trình tạo phần tử giao diện trong form như đã giới thiệu để tạo 1 đối tượng ImageList, đối tượng này sẽ chứa các ảnh bitmap được dùng cho các icon Toolbar, vị trí và kích thước của đối tượng ImageList không quan trọng vì nó sẽ bị ẩn khi chương trình chạy.

2.1 ấn phải chuột vào đối tượng ImageList rồi chọn mục Properties để hiển thị cửa sổ "Properties Page" của đối tượng ImageList.



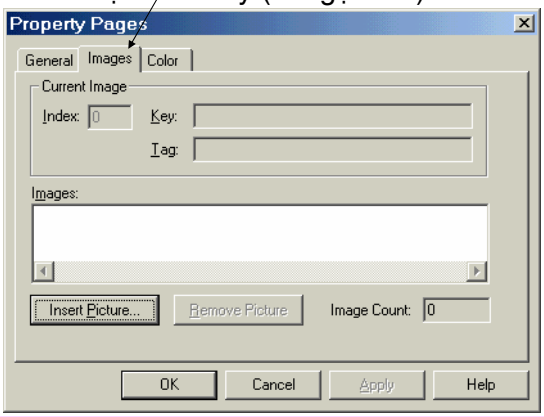
Qui trình tạo Toolbar của form (tt)

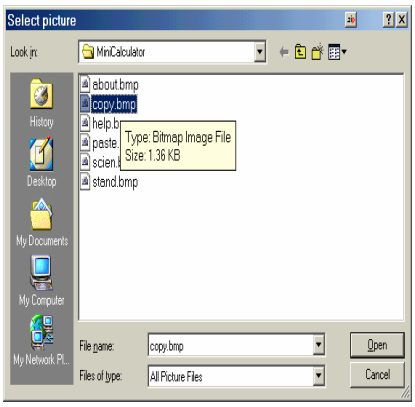
2.2 chọn tab General, chọn checkbox Custom rồi nhập kích thước của button Toolbar vào 2 field Height và Width.



Qui trình tạo Toolbar của form (tt)

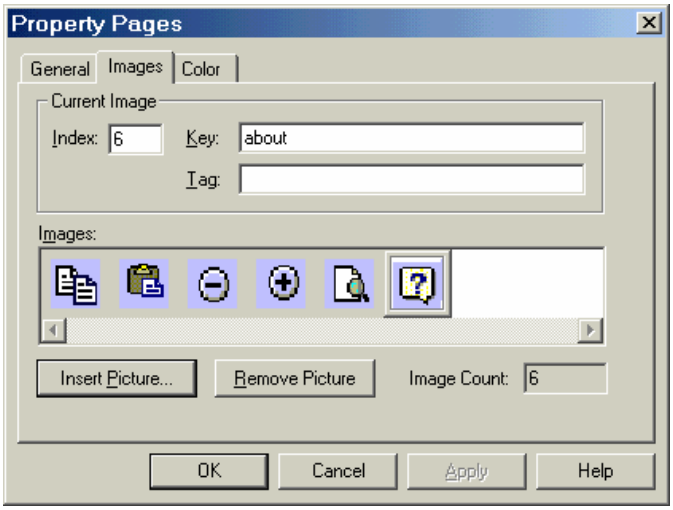
2.3 chọn tab Images rồi thêm từng ảnh button vào ImageList bằng trình tự : ấn Insert Picture, duyệt và chọn file image, nhập giá trị cho field Key. Để truy xuất ảnh button, ta dùng hoặc thuộc tính Index hoặc thuộc tính Key (tên gợi nhớ).





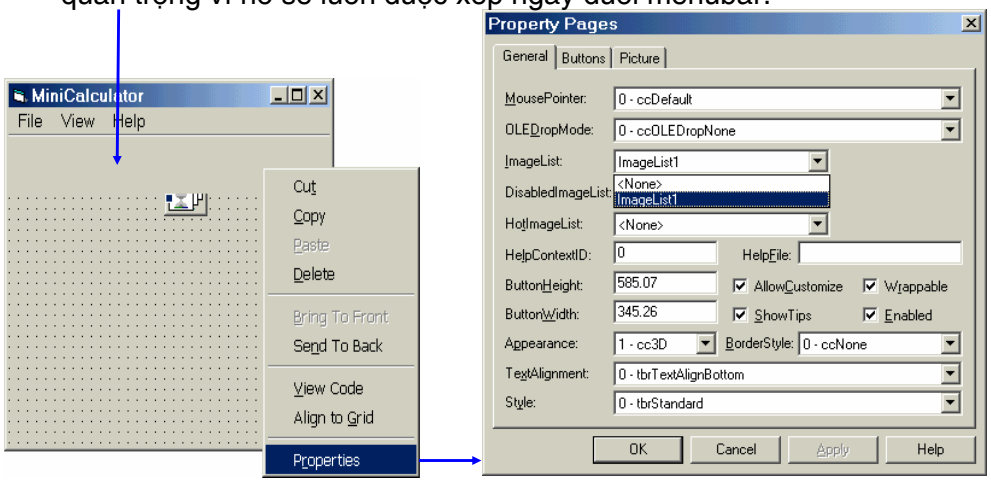
Qui trình tạo Toolbar của form (tt)

2.4 sau 6 lần insert icon vào ImageList, ta có kết quả như hình bên. Bạn có thể chọn lại từng icon để kiểm tra/hiệu chỉnh các thuộc tính của nó.



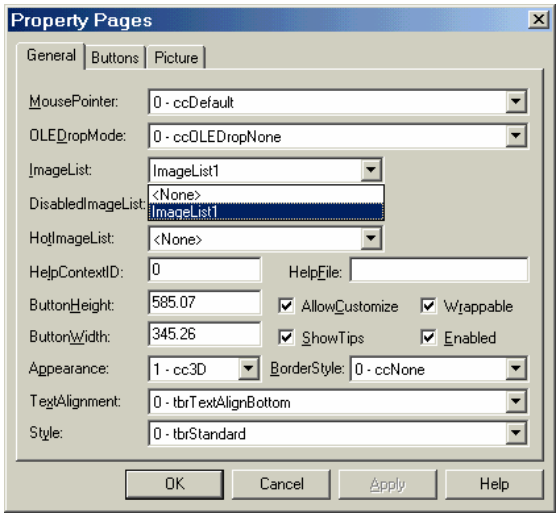
Qui trình tạo Toolbar của form (tt)

3. Tạo 1 đối tượng Toolbar, vị trí và kích thước của đối tượng này không quan trọng vì nó sẽ luôn được xếp ngay dưới menubar.



Qui trình tạo Toolbar của form (tt)

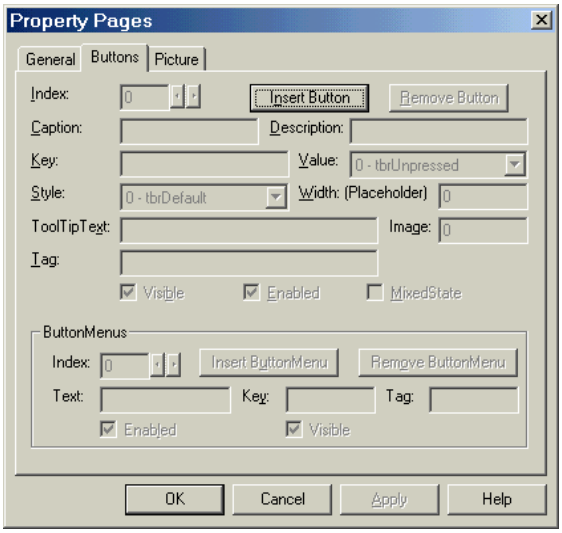
3.1 Mở cửa sổ thuộc tính của Toolbar, chọn tab General, chọn ImageList kết hợp với Toolbar trong listbox ImageList. Nếu muốn hình ảnh của từng icon khác nhau cho từng trạng thái (chưa chọn, được chọn, bị cấm), bạn phải tạo 2 ImageList khác : HotImageList (trạng thái được chọn) và DisableImageList (trạng thái bị cấm).



Qui trình tạo Toolbar của form (tt)

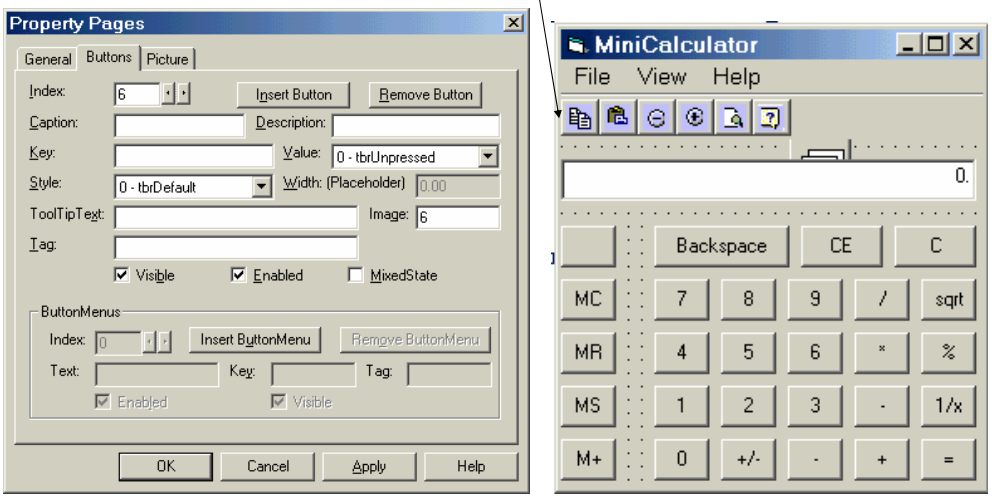
3.2 Chọn tab Buttons, thêm từng button vào Toolbar và thiết lập thuộc tính của nó bằng trình tự các hoạt động sau :

- ấn Insert Button để thêm button mới,
- nhập giá trị thuộc tính "Key",
- nhập chỉ số "Images" trong ImageList được dùng cho button,
- nhập trị cho thuộc tính "ToolTipText"...



Qui trình tạo Toolbar của form (tt)

3.3 sau khi thêm 6 button vào Toolbar thì Toolbar có dạng sau :



Ghi chú về qui trình tạo giao diện

- Thường thì form giao diện như MiniCalculator không cần chứa menubar và Toolbar. Giao diện dạng này được gọi là Dialog based.
- Còn 2 dạng giao diện ứng dụng phổ biến khác là :
  - SDI (Single Document Interface) : cửa sổ của chương trình gồm 1 menubar, 1 hay nhiều Toolbar và 1 cửa sổ làm việc duy nhất. Từng thời điểm, cửa sổ làm việc này sẽ cho phép hiển thị/hiệu chỉnh 1 document của ứng dụng.
  - và MDI (Multiple Document Interface) : cửa sổ của chương trình gồm 1 menubar, 1 hay nhiều Toolbar và n cửa sổ làm việc khác nhau, mỗi cửa sổ làm việc sẽ cho phép hiển thị/hiệu chỉnh 1 document của ứng dụng.

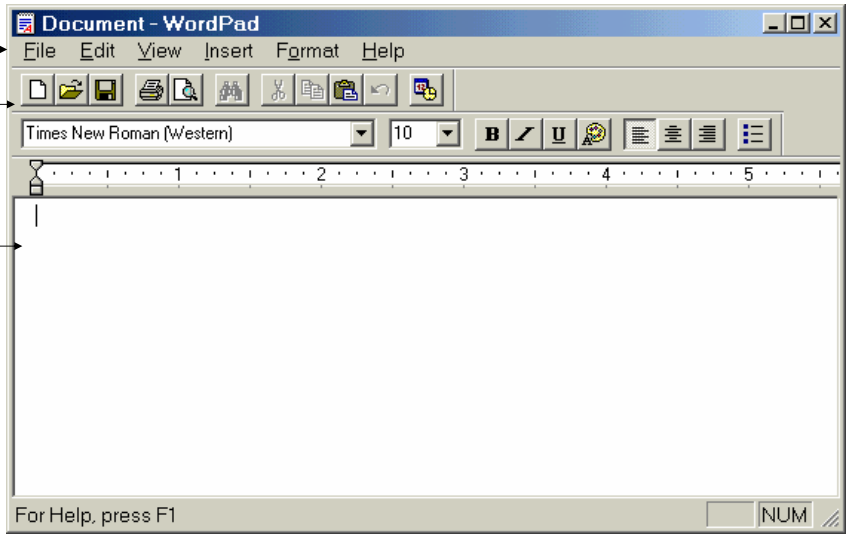
Giao diện SDI (Single Document Interface)

Menubar

Toolbar

Working region

(tại từng thời điểm chỉ có 1 để xử lý 1 document của ứng dụng)



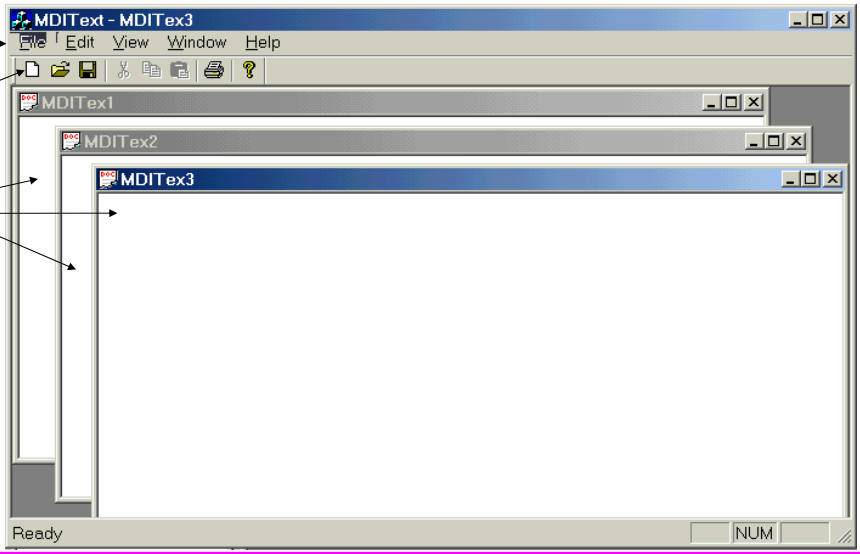
Giao diện MDI (Multiple Document Interface)

Menubar

Toolbar

Working region

(tại từng thời điểm có thể mở n cửa sổ để xử lý n document khác nhau)

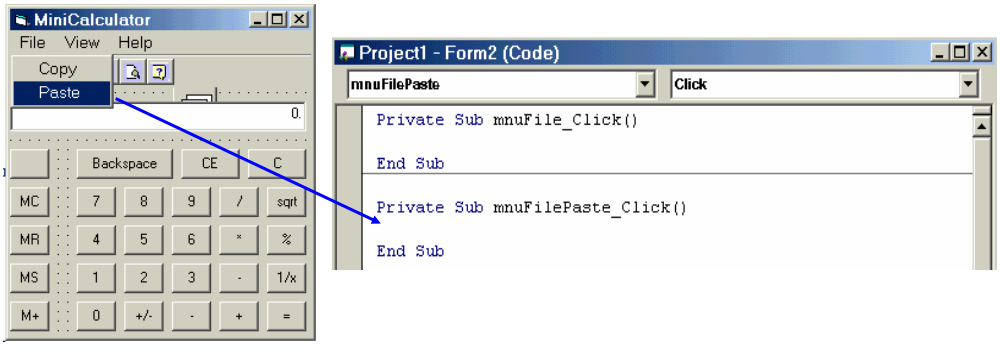


Ghi chú về qui trình tạo giao diện (tt)

- Để tạo giao diện trực quan của các ứng dụng SDI và MDI được dễ dàng, VB cung cấp cho người dùng 1 dạng Project tên là "VB Application Wizard".
- Chọn dạng Project này khi tạo Project, VB sẽ hướng dẫn người dùng tạo ra các phần tử giao diện rất dễ dàng, trong đó 2 đối tượng cơ bản là menubar và Toolbar được VB tạo tự động, người lập trình chỉ cần hiệu chỉnh lại theo yêu cầu riêng.

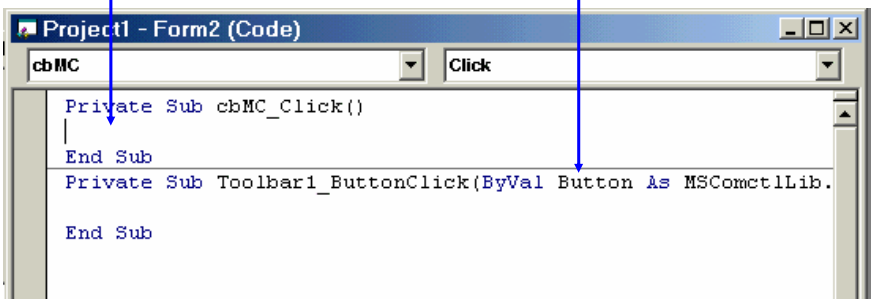
4.6 Tạo thủ tục xử lý sự kiện cho các đối tượng giao diện

- Sau khi đã thiết kế trực quan giao diện của ứng dụng theo yêu cầu, bạn sẽ tạo các thủ tục xử lý sự kiện cần thiết cho từng đối tượng giao diện.
- Để tạo thủ tục xử lý cho 1 option trong menu, bạn chọn menu tương ứng, dời chuột về option cần tạo thủ tục rồi chọn nó, template của thủ tục sẽ được tạo ra.
- Các chương tới sẽ giới thiệu cú pháp VB để bạn có thể viết code cho thủ tục.



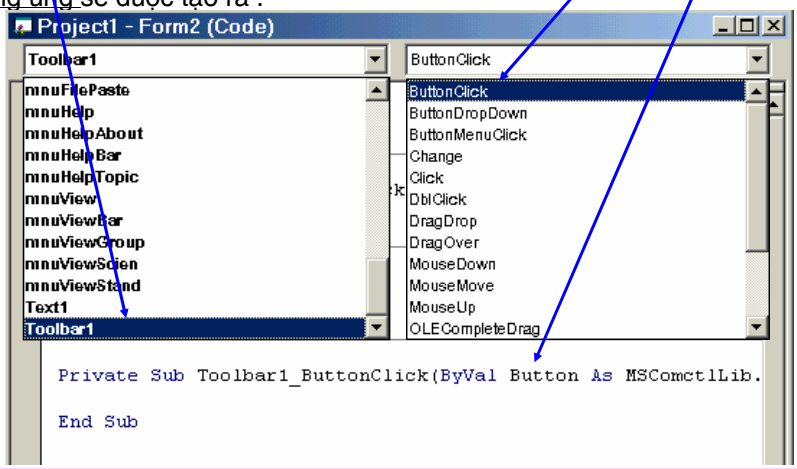
Tạo thủ tục xử lý sự kiện cho các đối tượng giao diện (tt)

- Để tạo thủ tục xử lý cho 1 button trong Toolbar, bạn ấn kép chuột vào button đó, template của thủ tục sẽ được tạo ra (lưu ý chỉ có 1 thủ tục xử lý cho tất cả các button trong 1 Toolbar, thủ tục này sẽ dựa vào thuộc tính Button.Key để biết button nào đã được chọn).
- Để tạo thủ tục xử lý cho 1 command button, bạn ấn kép chuột vào command button đó, template của thụ tục sẽ được tạo ra :



Tạo thủ tục xử lý sự kiện cho các đối tượng giao diện (tt)

- Cách tổng quát để tạo thủ tục xử lý sự kiện là mở cửa sổ code (View.Code), chọn tên đối tượng trong danh sách rồi chọn sự kiện cần xử lý, thủ tục xử lý tương ứng sẽ được tạo ra :



MÔN TIN HỌC

Chương 5

CÁC KIỂU DỮ LIỆU TRONG VB

5.1 Biến dữ liệu & định nghĩa biến
5.2 Các kiểu dữ liệu định sẵn của VB 6.0.
5.3 Việc dùng & tạo class đối tượng
5.4 Các tính chất chính yếu của biến dữ liệu
5.5 Hằng gợi nhớ

5.1 Biến dữ liệu

- Mỗi ứng dụng thường xử lý nhiều dữ liệu, ta dùng khái niệm "biến" để lưu trữ dữ liệu trong bộ nhớ máy tính, mỗi biến lưu trữ 1 dữ liệu của chương trình.
- Mặc dù VB không đòi hỏi, nhưng ta nên định nghĩa rõ ràng từng biến trước khi truy xuất nó để code của chương trình được trong sáng, dễ hiểu, dễ bảo trì và phát triển.
- Định nghĩa 1 biến là :
  - định nghĩa tên nhận dạng cho biến,
  - kết hợp kiểu với biến để xác định cấu trúc dữ liệu của biến,
  - định nghĩa tầm vực truy xuất biến.
- Cú pháp đơn giản của lệnh định nghĩa biến :

  [Static|Public|Private|Dim] *AVariable* **As** *Type*
- tại từng thời điểm, biến chứa 1 giá trị (nội dung) cụ thể. Theo thời gian nội dung của biến sẽ bị thay đổi theo tính chất xử lý của code.

Định nghĩa tên biến

- Cách đặt tên cho 1 biến (hay cho bất kỳ phần tử trong chương trình):
  - Tên biến có thể dài đến 255 ký tự,
  - Ký tự đầu tiên phải là một ký tự chữ (letter),
  - Các ký tự tiếp theo có thể là các ký tự chữ (letter), ký số (digit), dấu gạch dưới,
  - Tên biến không được chứa các ký tự đặc biệt như các ký tự : ^, &,), (,%, $, #, @, !, ~, +, -, *, …
  - VB không phân biệt chữ HOA hay chữ thường trong tên biến.

  Ví dụ: Tên biến hợp lệ — Tên biến không hợp lệ
  + Base1_ball — + Base.1 *:* vì có dấu chấm
  + ThisIsLongButOk — + Base&1 : vì có dấu &
  + 1Base_Ball : ký tự đầu là 1 số
- Nên chọn tên biến ngắn gọn nhưng thể hiện rõ ý nghĩa. Ví dụ: Ta muốn có một biến để lưu hệ số lãi suất ngân hàng (Interest Rate), ta nên dùng tên biến là: InterestRate hoặc Irate chứ không nên dùng tên biến là IR…

Định nghĩa tên biến (tt)

- Với ví dụ ở trước, dòng l*ệnh* sau đây:

  IterestRateEarned = Total*InterestRate

  sẽ dễ hiểu hơn dòng l*ệnh*

  IE = T*IR
- Khi viết tên biến ta nên viết hoa chữ đầu tiên của một từ có ý nghĩa.

  Ví dụ : InterestRate sẽ dễ đọc hơn interestrate hay iNTERestRaTe…
- Không được dùng tên biến trùng với các từ khoá như : Print, Sub, End… (từ khóa là những từ mà ngôn ngữ VB đã dùng cho những thành phần xác định của ngôn ngữ)

5.2 Các kiểu dữ liệu cơ bản định sẵn của VB

Byte : 1 byte, 0 to 255

Boolean : 2 bytes, **True** or **False**

Integer : 2 bytes, -32,768 to 32,767

Long (long integer) : 4 bytes

-2,147,483,648 to 2,147,483,647

Single (single-precision floating-point) :4 bytes

-3.402823E38 to -1.401298E-45 ;1.401298E-45 to 3.402823E38

Double (double-precision floating-point) : 8 bytes

-1.79769313486231E308 to -4.94065645841247E-324;
4.94065645841247E-324 to 1.79769313486232E308

Currency (scaled integer) : 8 bytes

-922,337,203,685,477.5808 to 922,337,203,685,477.5807

Các kiểu dữ liệu cơ bản định sẵn của VB (tt)

Decimal : 14 bytes

+/-79,228,162,514,264,337,593,543,950,335 (không có dấu chấm thập phân)
+/-7.9228162514264337593543950335 (có 28 ký số bên phải dấu chấm)
+/-0.0000000000000000000000000001 là số khác 0 nhỏ nhất.

Date : 8 bytes; từ January 1, 100 tới December 31, 9999

Object : 4 bytes; chứa tham khảo đến bất kỳ đối tượng nào

String (variable-length) : 10 bytes + độ dài của chuỗi

0 tới 2G ký tự

String*n (fixed-length) : chuỗi có độ dài cố định

1 tới 65,400 ký tự

Variant (with numbers) : 16 bytes

Bất kỳ số nào nằm trong phạm vi kiểu Double

Variant (with characters) : 22 bytes + độ dài chuỗi

cùng độ dài như kiểu String

User-defined (using Type) : kiểu do người dùng định nghĩa (record)

gồm nhiều field, mỗi field có kiểu riêng theo yêu cầu.

Các kiểu dữ liệu cơ bản định sẵn của VB (tt)

Array : dãy nhiều phần tử có cấu trúc dữ liệu đồng nhất, mỗi phần tử được truy xuất độc lập nhờ chỉ số của nó trong dãy.

Ví dụ : Dim vector(10) As Integer

định nghĩa biến vector là 1 dãy gồm 10 phần tử nguyên, vector(i) là tên nhận dạng của phần tử thứ i của dãy này.

Ngoài các kiểu dữ liệu định sẵn, VB còn cung cấp cho người lập trình 1 phương tiện để họ có thể định nghĩa bất kỳ kiểu dữ liệu chưa cung cấp sẵn nhưng lại cần thiết cho ứng dụng của họ, ta gọi các kiểu này là **kiểu do người dùng định nghĩa**. Thí dụ sau đây là phát biểu định nghĩa kiểu miêu tả các thông tin chính về máy tính cá nhân :

```
Type SystemInfo
    CPU As Variant
    Memory As Long
    DiskDrives(25) As String ' Fixed-size array.
    VideoColors As Integer
    Cost As Currency
    PurchaseDate As Variant
End Type
```


Đặc tính chi tiết về kiểu String

Kiểu String (chuỗi ký tự) :

- String là kiểu dữ liệu được dùng để lưu trữ chuỗi các ký tự (độ dài bất kỳ)
- Giá trị chuỗi ký tự được đặt trong cặp dấu nháy kép (vd : "Môn Tin học")
- Trên lý thuyết, một biến thuộc kiểu String có thể lưu trữ được đến 2 tỷ ký tự nhưng trong thực tế, độ dài của chuỗi bị hạn chế theo dung lượng bộ nhớ của máy tính.
- Có thể thực hiện được các phép toán nối kết chuỗi (+,&) trên các chuỗi ký tự và có khá nhiều hàm xử lý chuỗi có sẵn.
- Có thể định nghĩa một biến thuộc kiểu String như sau :

  Dim *AStringVariable* **As String**
  Dim *AStringVariable* **As String*100**
  Hay **Dim** *AStringVariable$*

  Tiếp vĩ ngữ $ đi sau tên biến dùng để khai báo một biến thuộc kiểu String (nhưng ta không nên dùng cách này vì tối nghĩa, khó bảo trì).

Đặc tính chi tiết về kiểu Integer

Kiểu Integer (Số nguyên) :

- Integer là kiểu dữ liệu được dùng để lưu trữ các số nguyên ngắn nằm trong khoảng từ - 32768 đến 32767.
- Số nguyên được lưu trữ trong bộ nhớ bằng 2 byte.
- Có thể thực hiện được các phép toán số học (như +,-,*,/,...) trên các dữ liệu thuộc kiểu Integer.
- Khai báo một biến thuộc kiểu Integer như sau :

  Dim *AnIntegerVariable* **As Integer**

  Hay **Dim** *AnIntegerVariable%*

  Tiếp vĩ ngữ % đi sau tên biến được dùng để khai báo một biến thuộc kiểu Integer.

Vd:
```
Dim Age As Integer
...
Age = 24
```


Đặc tính chi tiết về kiểu Long

Kiểu Long (Số nguyên dài) :

- Dùng để lưu trữ các số nguyên lớn nằm trong khoảng từ :

  -2,147,483,648 đến 2,147,483,647
- Số nguyên dài được lưu trữ trong bộ nhớ bằng 4 byte.
- Có thể thực hiện được các phép toán số học (như +,-,*,/,...) trên các dữ liệu thuộc kiểu Long.
- Khai báo một biến thuộc kiểu Long như sau :

  Dim *ALongIntegerVariable* **As Long**

  Hay **Dim** *AnIntegerVariable&*

  Tiếp vĩ ngữ & đi sau tên biến được dùng để khai báo một biến thuộc kiểu Long.

Vd:
```
Dim EarthAge As Long
...
EarthAge = 3276979
```

Đặc tính chi tiết về kiểu Byte

Kiểu Byte (Số nguyên dương nhỏ) :

- Dùng để lưu trữ các số nguyên không âm nằm trong tầm trị từ : 0 đến 255
- Chiếm 1 byte trong bộ nhớ.
- Có thể thực hiện được các phép toán số học (như +,-,*,/,...) trên các dữ liệu thuộc kiểu Byte.
- Kiểu dữ liệu này rất hiệu dụng trong việc lưu trữ các số nguyên nhỏ vì nó chiếm ít bộ nhớ, tốc độ xử lý nhanh.

Ví dụ: Khi cần lưu tuổi của một người chúng ta không nên dùng kiểu Integer hay Long mà nên dùng kiểu Byte vì tuổi của một người luôn là một số không âm, có giá trị tối đa thường nhỏ hơn 120.

Đặc tính chi tiết về kiểu Boolean

Kiểu Byte (tt) :

- Khai báo cho một biến thuộc kiểu Byte như sau :
  Dim *AByteVariable* **As Byte** 'không có tiếp vĩ ngữ

Ví dụ:

```
Dim Age As Byte
...
Age = 100
```

Kiểu Boolean (giá trị luận lý) :

- Là kiểu dữ liệu lưu trữ hai giá trị luận lý True/False.
- Được lưu trữ trong máy tính bằng 2 byte.
- Khai báo
  Dim *ABooleanVariable* **As Boolean**

Đặc tính chi tiết về kiểu Single

Kiểu Single (số thực có độ chính xác đơn - Single Precision) :

- Lưu trữ các số thực có độ chính xác đơn (gần đúng với giá trị gốc với độ chính xác ở mức 7 chữ số)

  Ví dụ số 1234.567 thì ký số 7 (bên phải nhất) có thể không chính xác.
- Kiểu dữ liệu này chiếm 4 byte trong bộ nhớ và miêu tả các giá trị trong phạm vi :

  Từ -3.402823E38 đến -1.401298E-45 cho các giá trị âm

  và từ 1.401298E-45 đến 3.402823E38 cho các giá trị dương
- Có thể thực hiện các phép toán số học trên kiểu dữ liệu này nhưng thường chậm hơn so với các biến thuộc kiểu Integer hay Long. Do cách chứa số Single chỉ ở mức gần đúng nên các phép toán trên các dữ liệu thuộc kiểu này sẽ tạo kết quả gần đúng (nhưng đủ dùng trong đại đa số yêu cầu thực tế).
- Khai báo như sau :

  Dim *ASingleVariable* **As Single**

Hay **Dim** *ASingleVariable!* 'Tiếp vĩ ngữ là dấu !

Ví dụ:

```
Dim InterestRate As Single, Earned!, Total As Single
Earned = InterestRate * Total
```


Đặc tính chi tiết về kiểu Double

Kiểu Double (số thực có độ chính xác kép - Double Precision) :

- Lưu trữ các số thực có độ chính xác kép (gần đúng với giá trị gốc với độ chính xác ở mức 16 chữ số).

  Ví dụ số 1234.57890123456 thì ký số bên phải nhất (6) có thể không chính xác.
- Kiểu dữ liệu này chiếm 8 byte trong bộ nhớ và miêu tả các giá trị trong phạm vi :

  -1.797693234862232E308 đến -4.94065645841247E-324

  và 4.94065645841247E-324 đến 1.797693234862232E308
- Có thể thực hiện các phép toán số học trên kiểu dữ liệu này nhưng rất chậm (chậm hơn cả kiểu Single). Do cách chứa số Double chỉ ở mức gần đúng nên các phép toán trên các dữ liệu thuộc kiểu này sẽ tạo kết quả gần đúng (nhưng quá đủ dùng trong đại đa số yêu cầu thực tế).
- Khai báo như sau :

  Dim *ADoubleVariable* **As Double**

Hay **Dim** *ADoubleVariable#* ' Tiếp vĩ ngữ là dấu #

Ví dụ:

```
Dim InterestRate As Double, Earned#, Total As Double
Earned = InterestRate * Total
```

Đặc tính chi tiết về kiểu Currency

Kiểu Currency (Tiền Tệ)

- Kiểu Currency được dùng để lưu các dữ liệu thuộc kiểu tiền tệ (số lượng tiền).
- Được lưu trữ trong bộ nhớ bằng 8 byte.
- Có thể có 4 chữ số ở bên phải dấu chấm thập phân và 15 chữ số ở bên trái dấu thập phân.
- Có tầm trị: - 922337203685477.5808 đến 922337203685477.5807
- Có thể thực hiện được các phép toán số học trên kiểu dữ liệu này nhưng tốc độ xử lý rất chậm như đối với các số thực có độ chính xác kép, song nó là kiểu dữ liệu ưa dùng cho các phép tính tài chính.
- Khai báo: **Dim** *ACurrencyVariable* **As Currency**
  Hay **Dim** *ACurrencyVariable*@ 'Tiếp vĩ ngữ là dấu @

Đặc tính chi tiết về kiểu Date

Kiểu **Date** (Ngày tháng)

- Dùng để lưu trữ các dữ liệu thuộc kiểu ngày giờ cho bất kỳ thời điểm nào từ 0h00 ngày 01/01/100 đến 0h00 ngày 31/12/9999.
- Kiểu Date được lưu trữ trong máy tính bằng 8 bytes.
- Dữ liệu thuộc kiểu Date phải được bao bởi cặp dấu # ở hai đầu.

Ví dụ: Millenium = #January 1, 2000#
Millenium = #Jan 1, 2000#
Millenium = #1/1/ 2000#

- Nếu ta chưa gán trị cho biến thuộc kiểu Date thì VB mặc nhận đó là lúc 0:0:0 cùng ngày.
- Có thể dùng dạng thức AM/PM hay dạng 24 giờ để biểu diễn cho giá trị giờ

Ví dụ: PreMillenium = #December 31, 1999 11:59:59PM#
hay PreMillenium = #December 31, 1999 23:59:59#

- Khai báo một biến thuộc kiểu Date như sau:
  Dim *ADateVariable* **As Date** 'Không có tiếp vĩ ngữ

Ví dụ : Dim PreMillenium As Date

Đặc tính chi tiết về kiểu Variant

Kiểu **Variant** (Kiểu dữ liệu biến đổi)

- Kiểu dữ liệu này được thiết kế để lưu mọi dữ liệu thuộc kiểu định sẵn của VB. Ví dụ như : Date, String, Double, Integer…
- Nếu không khai báo kiểu rõ ràng cho 1 biến thì biến này sẽ được hiểu là thuộc kiểu này.
- VB sẽ chuyển đổi dữ liệu thuộc kiểu Variant thành một kiểu dữ liệu khác cho phù hợp (khi gán dữ liệu,...).

Ví dụ : String ← Variant,Integer ← Variant,Date ← Variant

- Tuy nhiên việc chuyển đổi kiểu như trên sẽ dẫn đến nhiều lỗi không lường trước được.
- Dùng kiểu Variant thay cho một kiểu cụ thể sẽ làm chậm tốc độ xử lý của chương trình do phải tốn thời gian chuyển đổi và tốn nhiều bộ nhớ hơn.
- Khai báo một biến thuộc kiểu Variant như sau :

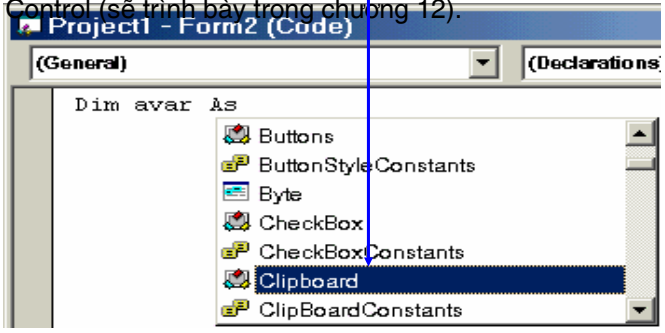
  Dim *AVariantVariable* **As Variant**

Hay **Dim** *AVariantVariable* 'Mặc nhiên thuộc kiểu Variant

Ví dụ: Dim x, y, z As Integer 'x,y là kiểu Variant

5.3 Việc dùng các class đối tượng có sẵn

- Như chúng ta đã trình bày trong chương 3, VB hỗ trợ việc lập trình OOP ở 1 mức độ nhất định :
  - VB cung cấp 1 số class đối tượng, người lập trình có thể dùng chúng ở bất kỳ project ứng dụng nào, ta có thể nói rằng tên của các class định sẵn này cũng là kiểu định sẵn của VB.
  - VB cho phép dùng các class đối tượng được người khác viết thông qua công nghệ COM, ActiveX Control (sẽ trình bày trong chương 12).
  - VB cho người lập trình định nghĩa các class mới ngay trong project phần mềm của họ nhờ khái niệm "class module" (sẽ được trình bày chi tiết trong chương 6).



Việc dùng các class đối tượng có sẵn (tt)

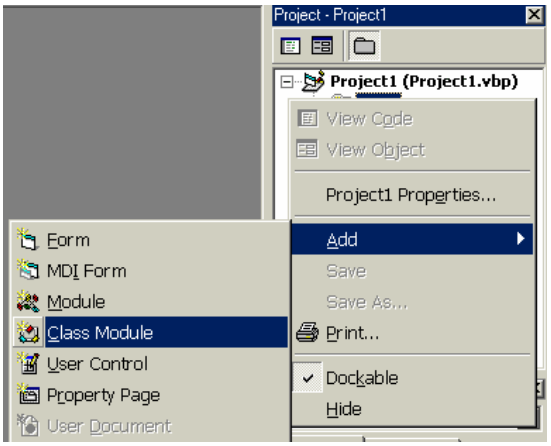
- Dù ở dạng nào, mỗi class đều có tên nhận dạng, tên này chính là tên kiểu và được dùng trong phát biểu định nghĩa biến đối tượng :

```
Dim pclipbd As Clipboard
...
Set pclipbd = New Clipboard
```

- Lưu ý rằng biến thuộc kiểu class không chứa trực tiếp đối tượng, nó chỉ là tham khảo (phương tiện truy xuất) đến đối tượng. Do đó trước khi dùng biến đối tượng, nhất thiết phải tạo đối tượng (thường thông qua hàm New) để nhận tham khảo đến đối tượng rồi gán tham khảo này vào biến như thí dụ trên.

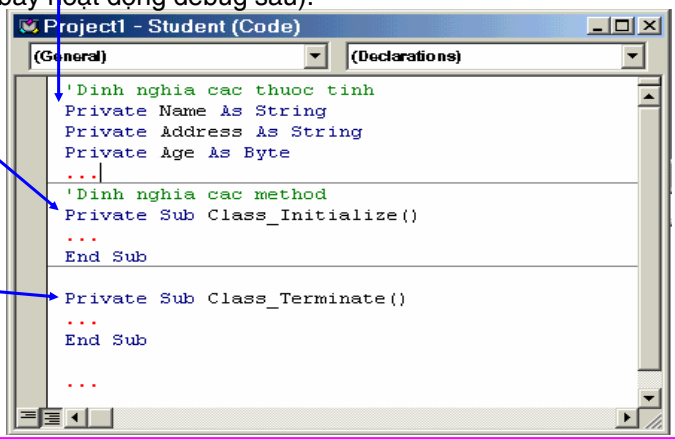
Qui trình tạo 1 class mới để dùng trong project

- Ấn chuột phải trong cửa sổ Project, dời chuột tới mục "Add" rồi chọn mục "Class Module" trong danh sách.
- Khi cửa sổ "Add class module" hiển thị, chọn icon "Class module" rồi ấn "Open" để tạo ra class mới.
- thiết lập tên class module cho phù hợp (trong cửa sổ thuộc tính), tên này chính là tên kiểu được dùng trong phát biểu định nghĩa biến đối tượng.
- ấn kép chuột vào mục tên class mới tạo ra để hiển thị cửa sổ code đặc tả cho class đó.



Cửa sổ đặc tả class

- định nghĩa từng thuộc tính dữ liệu và từng method của class theo cú pháp VB (sẽ được trình bày sau).
- debug từng method của class xem nó hoạt động đúng chức năng mong muốn trước khi dùng (sẽ trình bày hoạt động debug sau).
- mỗi class có 2 method đặc biệt :
  - Private Sub Class_Initialize() : miêu tả các hành động khi đối tượng vừa được tạo ra.
  - Private Sub Class_Terminate() : miêu tả các hành động khi đối tượng sắp sửa bị xóa.



5.4 Đặt tên biến theo cách "ký hiệu Hungarian"

Ký hiệu Hungarian

- Một số lập trình viên thường thích bổ sung thêm một tiếp đầu ngữ vào tên biến để nêu rõ kiểu của biến, nhờ đó tăng độ dễ đọc cho chương trình.

Ví dụ: sngInterestRate : Biến có độ chính xác đơn (Single)
intCount: Biến thuộc kiểu Integer
strName: Biến thuộc kiểu String

- Qui ước trên gọi là ký hiệu Hungarian (do Charles Simonge, một lập trình viên gốc Hungari của Miscrosoft) đề xướng. Lưu ý rằng bạn vẫn phải khai báo kiểu cho các biến trên và kiểu phải tương thích với tiếp đầu ngữ đã dùng với tên biến.
- Một số kiểu và tiếp đầu ngữ tương ứng :

String	*str*	Integer	*int*	Single	*sng*	Currency	*cur*
Boolean	*bln*	Long	*lng*	Double	*dbl*	Variant	*vnt*

Các lưu ý về biến

- Trong một ngữ cảnh (trong 1 chương trình con, trong 1 module, cấp toàn cục), không thể dùng hai biến cùng tên (VB không phân biệt chữ HOA hay chữ thường).
- Tên biến là phần tên, không kể đến tiếp vĩ ngữ miêu tả kiểu kết hợp.
  Ví dụ: Biến a% và biến a! là trùng nhau, VB sẽ đưa ra thông báo lỗi: “Duplicate Definition”
- Sau khi định nghĩa biến, VB sẽ khởi động trị ban đầu cho biến đó. Ví dụ :
  - Biến thuộc kiểu Variant có giá trị ngầm định là “Empty” (rỗng), giá trị “Empty” sẽ biến mất khi ta gán cho biến một giá trị cụ thể.
  - Biến chuỗi có giá trị ngầm định là chuỗI rỗng "" (hai dấu nháy liền nhau).
  - Biến số có giá trị ngầm định là 0.
- Không nên tin vào trị ngầm định của biến, phải gán giá trị cho biến trước khi dùng chúng.
- 1 biến tương ứng với 1 vùng nhớ, do đó khi gán một giá trị cho biến, giá trị cũ của biến sẽ bị mất đi.

Tầm vực truy xuất biến

- Tầm vực của một biến là tập các lệnh được phép truy xuất biến đó.
- Như được miêu tả trong silde 7 (chương 1), VB cho phép 3 cấp độ tầm vực sau :
  - **cục bộ trong thủ tục** : bất kỳ lệnh nào trong thủ tục đều có thể truy xuất được biến cục bộ trong thủ tục đó.

    ```
    Private Sub Command1_Click()
      Dim strGreeting As String 'Khai báo cục bộ
      ...
    End Sub
    ```
  - **cục bộ trong module** : bất kỳ lệnh nào trong module đều có thể truy xuất được biến cục bộ trong module đó.

    ```
    Private strAddr As String 'biến cục bộ trong module
    Public strName As String 'biến toàn cục
    ```
  - **toàn cục** : bất kỳ lệnh nào trong chương trình cũng có thể truy xuất được biến toàn cục.
- Trong một ngữ cảnh (cùng 1 thủ tục, cùng 1 module, hay cấp toàn cục), không thể dùng hai biến cùng tên (VB không phân biệt chữ HOA hay chữ thường).

Thời gian sống của biến

- Biến là 1 thực thể nên cũng có thời gian sống hữu hạn, thời gian sống của biến thường phụ thuộc vào tầm vực của biến đó :
  - **biến cục bộ trong thủ tục** : được tạo ra lúc thủ tục được gọi và mất đi khi thủ tục kết thúc việc xử lý và điều khiển được trả về lệnh gọi.
  - **biến cục bộ trong module** : được tạo ra lúc module được tạo ra và mất đi khi module bị xóa.
    - Các (standard) modules có thời gian sống từ lúc chương trình chạy cho đến khi chương trình kết thúc.
    - Các đối tượng của class module hay form module được tạo ra khi có yêu cầu cụ thể. Tạo đối tượng nghĩa là tạo các thuộc tính của nó, các thuộc tính của đối tượng sẽ mất đi khi đối tượng bị xóa.
  - **biến toàn cục** : được tạo ra lúc chương trình bắt đầu chạy và chỉ mất đi khi chương trình kết thúc.

Thời gian sống của biến (tt)

- Muốn kéo dài thời gian sống của 1 biến, ta thường dùng 2 cách sau :
  - **nâng cấp tầm vực** : từ cục bộ trong thủ tục lên cục bộ trong module hay lên toàn cục... Cách này ít được dùng tường minh vì nó sẽ thay đổi tầm vực của biến. Để khắc phục điều này, VB cung cấp khái niệm "Static" kết hợp với biến : biến có thuộc tính "Static" sẽ tồn tại mãi và chỉ mất đi khi chương trình kết thúc bất chấp tầm vực của nó ra sao.

```
Private Sub Command1_Click()
  Dim strGreeting As String 'biến cục bộ
  Static strAddr As String
  'biến cục bộ có thời gian sống lâu dài theo ứng dụng.
  ...
End Sub
```

  - **ghi giá trị biến ra môi trường chứa tin bền vững** (file trên đĩa) trước khi biến bị xóa. Khi cần lại giá trị của biến này, ta đọc giá trị của nó từ file vào. Đây là phương pháp thông dụng để trao đổi dữ liệu giữa 2 ứng dụng khác nhau hay giữa 2 lần chạy khác nhau của cùng 1 ứng dụng.

5.5 Hằng gợi nhớ

- Như ta đã biết, ta định nghĩa biến để lưu trữ dữ liệu của chương trình. Ngay sau khi được định nghĩa, giá trị ban đầu của biến thường chưa được xác định tường minh, do đó ta phải thiết lập (gán) giá trị cho biến trước khi dùng lại trị của biến.
- Có 3 cách khác nhau để thiết lập (gán) giá trị cho 1 biến :
  - từ tương tác với người dùng, biến kết hợp với đối tượng giao diện tương ứng sẽ được gán giá trị mà người dùng nhập vào.
  - các tham số được truyền khi gọi thủ tục, khi hoàn thành code trong thủ tục sẽ gán trị vào tham số.
  - nhưng cách cơ bản và phổ biến nhất là dùng phát biểu gán với cú pháp sau :

    AVariable = *AExpression*

Hằng gợi nhớ

- Biểu thức đơn giản nhất là 1 giá trị, giá trị này có thể được dùng nhiều nơi trong chương trình.
- Hằng gợi nhớ (Constant) là khái niệm cho phép người lập trình kết hợp 1 tên gợi nhớ với 1 giá trị để khi cần dùng giá trị đó, ta không viết lại chi tiết cụ thể của trị mà chỉ dùng tên gợi nhớ.
- Cú pháp của phát biểu định nghĩa hằng gợi nhớ :

  Const *ConstName* = *Value*

  Ví dụ : Const PI = 3.1416
- Lợi ích của việc dùng hằng gợi nhớ :
  - Chương trình sẽ trong sáng, dễ đọc hơn, dẫn đến việc bảo trì, nâng cấp chương trình được thuận tiện hơn.
  - Tiết kiệm được bộ nhớ so với việc dùng biến.
  - Rút ngắn được các câu lệnh quá dài

MÔN TIN HỌC

Chương 6

CÁC LỆNH ĐỊNH NGHĨA & KHAI BÁO VB

6.1 Tổng quát về code của 1 ứng dụng VB

- Một project VB thường quản lý các thành phần cấu thành 1 ứng dụng VB.
- Trong 1 project VB có 3 loại phần tử có chứa code (do đó ta cần biết cú pháp VB để xây dựng các loại phần từ này) :
  - **class module** định nghĩa sự hiện thực của 1 lớp đối tượng có cấu trúc và hành vi giống nhau.
  - **form module** là trường hợp đặc biệt của class module, nó miêu tả sự hiện thực của 1 lớp đối tượng đặc biệt : một form giao diện.
  - **(standard) module** là đơn vị phần mềm nhỏ có 1 chức năng rõ ràng nào đó. Theo trường phái lập trình cấu trúc (cổ điển), ta dùng module để chia ứng dụng ra nhiều phần nhỏ dễ quản lý $\Rightarrow$ VB hỗ trợ cả 2 phương pháp lập trình : có cấu trúc và OOP.
- Ngoài 1 vài ngoại lệ nhỏ, tổ chức code cho 3 loại module trên hoàn toàn giống nhau : đó là danh sách nhiều lệnh VB phục vụ định nghĩa kiểu, hằng, biến và thủ tục trong module đó. Trong lệnh định nghĩa thủ tục, ta sẽ dùng các lệnh thực thi để miêu tả giải thuật của thủ tục.

Tổng quát về ngôn ngữ VB

- Để dễ tiếp cận ngôn ngữ VB, ta hãy nhìn lại ngôn ngữ tiếng Việt. Ta nói ngôn ngữ tiếng Việt định nghĩa 1 tập các từ có nghĩa cơ bản, các qui tắc ghép các từ cơ bản này lại để tạo thành đoạn câu (phrase), câu (sentence), đoạn văn (paragraph), bài văn (document) cùng ngữ nghĩa của các phần tử được tạo ra. Vì ngôn ngữ Việt là ngôn ngữ tự nhiên nên thường cho phép nhiều ngoại lệ trong việc xây dựng các phần tử.
- Ngôn ngữ lập trình VB cũng định nghĩa 1 tập các ký tự cơ bản (chưa có nghĩa), các qui tắc ghép các ký tự để tạo thành các từ có nghĩa (identifier), biểu thức (expression), câu lệnh (statement), thủ tục (Function, Sub, Property) cùng ngữ nghĩa của các phần tử được tạo ra. Vì ngôn ngữ VB là ngôn ngữ lập trình cho máy tính thực hiện nên sẽ không cho phép 1 ngoại lệ nào trong việc xây dựng các phần tử.
- Nghiên cứu ngôn ngữ lập trình là học để nhớ rõ các ký tự cơ bản của ngôn ngữ, các qui tắc để tạo danh hiệu, biểu thức, các qui tắc để viết các câu lệnh... cùng ngữ nghĩa của chúng ⇒ rất giống với việc học 1 ngôn ngữ tự nhiên : Anh, Pháp, Nhật,...

Các ký hiệu cơ bản của ngôn ngữ VB

- Về nguyên tắc, VB cho phép dùng hầu hết các ký tự mà bạn có thể nhập từ bàn phím, trong đó các ký tự chữ và số được dùng chủ yếu.
- Qui tắc cấu tạo 1 danh hiệu đã được trình bày ở Slide 113 (chương 5). 1 danh hiệu có thể được dùng để đặt tên cho biến, hằng gợi nhớ, Function, Sub, Property, form, class module, module,... và ngữ nghĩa của từng danh hiệu là do sự qui định của người lập trình.
- Qui tắc xây dựng 1 biểu thức sẽ được trình bày trong chương 7.
- Có nhiều loại câu lệnh VB khác nhau, qui tắc xây dựng 1 câu lệnh phụ thuộc vào loại câu lệnh cụ thể ⇒ ta phải nghiên cứu từng loại câu lệnh và qui tắc cấu thành nó, nhưng may mắn số lượng loại câu lệnh VB là không nhiều (dưới 20 loại).
- Các câu lệnh được chia làm 2 nhóm chính :
  - **các lệnh định nghĩa** : xác định 1 hành động nào đó tại thời điểm dịch.
  - và **các lệnh thực thi** : xác định 1 hành động nào đó tại thời điểm thực thi.

6.2 Chú thích trong chương trình

- Các lệnh định nghĩa và các lệnh thực thi mà ta vừa trình bày là để máy xử lý, chúng tuân thủ các cú pháp cụ thể mà ta sẽ trình bày sau. Nhưng ý tưởng chung là con người rất khó đọc và hiểu chúng.
- Để trợ giúp cho người đọc và hiểu các lệnh VB trong chương trình, VB còn cung cấp 1 lệnh đặc biệt : lệnh chú thích. Đây là lệnh mà máy sẽ bỏ qua (vì máy sẽ không thể hiểu nổi ý nghĩa được miêu tả trong lệnh này), tuy nhiên lệnh này cho phép người lập trình dùng ngôn ngữ tự nhiên để chú thích ý nghĩa của các lệnh VB khác hầu giúp chính họ hay người khác dễ dàng hiểu chương trình.
- Cú pháp của lệnh chú thích rất đơn giản : chỉ qui định bắt đầu lệnh bằng ký tự ' và có thể được viết trên 1 hàng riêng biệt hay đi sau lệnh hiện hành.

Ví dụ :

```
Private Sub cmdCE_Click()
' hàm xử lý biến cố khi ấn nút CE (Clear Entry)
        dblDispValue = 0
        blnFpoint = False
        bytPosDigit = 0
        txtDisplay.Text = ".0"      ' bắt đầu hiển thị .0 lên Display
End Sub
```


Chú thích trong chương trình (tt)

- Việc dùng chú thích trong chương trình là sự dung hòa giữa 2 thái cực : lạm dụng và không bao giờ dùng. Thường ta nên dùng chú thích ở những vị trí sau :
  - ở đầu của mỗi thủ tục để miêu tả chức năng của thủ tục đó, dữ liệu nhập vào thủ tục và dữ liệu trả về từ thủ tục.
  - ở các đoạn code miêu tả giải thuật phức tạp để ghi chú đoạn code này hiện thực giải thuật nào trong lý thuyết đã học.
  - ở hàng lệnh có hiệu ứng đặc biệt...

6.3 Các lệnh định nghĩa

- 1 module VB (form, class, standard) gồm 2 loại phần tử : thuộc tính dữ liệu và các method (thủ tục). Các lệnh định nghĩa cho phép ta định nghĩa tính chất của các thuộc tính dữ liệu, các lệnh thực thi cho phép ta miêu tả giải thuật thi hành của các method (thủ tục).
- 2 lệnh định nghĩa dữ liệu chủ yếu là lệnh định nghĩa biến và lệnh định nghĩa hằng, trong 2 lệnh này có sử dụng tên kiểu dữ liệu. Tên kiểu dữ liệu có thể là định sẵn, có thể do người lập trình tự đặt. Lệnh định nghĩa kiểu sẽ phục vụ việc định nghĩa kiểu mới của người lập trình.
- Để VB kiểm tra việc định nghĩa biến bắt buộc trong 1 module nào đó, ta dùng lệnh sau ở đầu module đó.

  `Option Explicit`
- Cú pháp định nghĩa hằng gợi nhớ cơ bản :

  `Const` *`AConst`* **`=`** *`Value`*

  Lưu ý ta dùng *chữ nghiêng* để miêu tả phần tử mà người lập trình tự xác định theo yêu cầu riêng (dĩ nhiên phải thỏa mãn qui tắc VB), chữ đậm miêu tả phần tử bắt buộc và người lập trình phải viết y như vậy trong lệnh của họ.

Qui tắc miêu tả các loại giá trị

- Giá trị luận lý : **True** | **False**.
- Giá trị thập phân nguyên : [**+|-**] [*decdigit*]+ (Vd. 125, -548)

  Lưu ý ta dùng | để miêu tả sự chọn lựa, [...] để miêu tả có từ 0 tới 1, [...]* để miêu tả có từ 0 tới n, [...]+ để miêu tả có từ 1 tới n (n>1).
- Giá trị thập lục phân nguyên : [**+|-**] **&H**[*hexdigit*]+ (&HFF)
- Giá trị bát phân nguyên : [**+|-**] **&O**[*ocdigit*]+ (&O77)
- Giá trị thập phân thực :

  [**+|-**] [*decdigit*]+ [**.**[*decdigit*]*] [**E** [**+|-**] [*decdigit*]+]

  3.14159, 0.31459E1,-83.1E-9,...
- Giá trị chuỗi : "Nguyen Van A"

  """Nguyen Van A"""

  Lưu ý dùng 2 dấu nháy kép liên tiếp để miêu tả 1 ký tự nháy kép trong giá trị chuỗi (cơ chế dùng Escape để giải quyết nhầm lẫn).

Qui tắc miêu tả các loại giá trị (tt)

- Giá trị ngày tháng (Date) : đã trình bày trong slide 125, ở đây ta chỉ nhắc lại cho có tính hệ thống.

Ví dụ: #January 1, 2000#
#Jan 1, 2000#
#1/1/ 2000#
#December 31, 1999 11:59:59PM#
#December 31, 1999 23:59:59#

- Giá trị ngày tháng luôn được đặt trong cặp dấu #....#.
- Có nhiều dạng thức khác nhau để miêu tả giờ trong ngày và miêu tả ngày/tháng/năm. Dạng thức miêu tả ngày dạng 2/1/2000 sẽ được phân giải theo thông số "locale" của Windows (dạng dd/mm/yyyy hay mm/dd/yyyy).

6.4 Phát biểu định nghĩa biến

- Cú pháp cơ bản của định nghĩa biến cục bộ trong function, Sub, Property :

  Dim *AVariable* [**As** *Type*]

  Static *AVariable* [**As** *Type*]

- Cú pháp cơ bản của định nghĩa biến cục bộ trong module (class, form, standard) :

  Private *AVariable* [**As** *Type*]

  Static *AVariable* [**As** *Type*]

- Cú pháp cơ bản của định nghĩa biến toàn cục :

  Public *AVariable* [**As** *Type*]

- Lưu ý hạn chế tối đa việc dùng biến toàn cục (trong OOP ta không cần dùng biến toàn cục).

Phát biểu định nghĩa biến (tt)

- Có thể dùng tiếp vĩ ngữ qui định kiểu (trong chương 5) thay thế cho tên kiểu. Nếu tên biến không có tiếp vĩ ngữ và không có phần tên kiểu trong lệnh định nghĩa biến thì biến thuộc kiểu Variant. Cho phép nhiều phát biểu định nghĩa biến trên 1 hàng lệnh (dùng dấu ',' để ngăn cách chúng).
- Nên đặt tên biến theo ký hiệu Hungarian và luôn miêu tả tên kiểu kết hợp với biến trong lệnh định nghĩa biến, nhờ vậy chương trình sẽ rất trong sáng, dễ hiểu và dễ phát triển.

Ví dụ :

Thay vì dùng lệnh sau :

```
Private DispValue#
```

để định nghĩa biến thực chính xác kép tên là "DispValue", ta nên dùng lệnh định nghĩa sau :

```
Private dblDispValue As Double
```


6.5 Phát biểu định nghĩa kiểu của người lập trình

- Nếu trong 1 module nào đó cần dữ liệu có cấu trúc đặc thù mà VB chưa cung cấp, người lập trình sẽ dùng phát biểu TYPE để định nghĩa kiểu này. Phát biểu này kết hợp tên kiểu (tự đặt) với 1 cấu trúc dữ liệu gồm nhiều field dữ liệu (do đó ta thường gọi kiểu này là kiểu record hay structure). Cú pháp như sau :

```
Type TypeName
[AfieldName As Type]+
End Type
```

Ví dụ :

```
Type SystemInfo
    CPU As Variant
    Memory As Long
    DiskDrives(25) As String        ' Fixed-size array.
    VideoColors As Integer
    Cost As Currency
    PurchaseDate As Variant
End Type
```

Phát biểu định nghĩa kiểu Array

- Nếu trong 1 module nào đó cần danh sách gồm nhiều dữ liệu có cấu trúc đồng nhất, ta sẽ dùng phát biểu định nghĩa kiểu array để miêu tả danh sách này. Cú pháp cơ bản như sau :

  Dim *varname*[**(**[*subscripts*]**)**] [**As** [**New**] *type*]

  trong đó subscripts là danh sách từ 1 đến n chiều cách nhau bằng dấu ',', mỗi chiều miêu tả phạm vi chỉ số các phần tử thuộc chiều đó ở dạng :

  [*lower* **To**] *upper*.

- Nếu chỉ số cận dưới của 1 chiều nào đó không được miêu tả thì VB chọn giá trị ngầm định (là 0 hay 1).
- Phát biểu định nghĩa giá trị cận dưới ngầm định có cú pháp :

  Option Base {**0|1**}

  Lưu ý dấu {..} miêu tả có 1 và chỉ 1 lần. Nếu không có phát biểu này thì VB chọn cận dưới là 0.

  Ví dụ :

  Dim vector(50) As Double 'vector có 51 phần từ từ 0 - 51

  Dim MyArray (1 to 100, 1 to 50) As Double

Phát biểu định nghĩa kiểu Array (tt)

Nếu số lượng phần tử của danh sách chưa biết tại thời điểm viết chương trình và chỉ biết tại thời điểm chạy, ta dùng 1 trong 2 cách sau :

- khai báo số lượng tĩnh tại thời điểm viết, cách này thường phí phạm bộ nhớ hay khai báo thiếu số lượng phần tử.
  - Thí dụ để giải hệ n phương trình tuyến tính, n ẩn số, ta có thể khai báo tĩnh ma trận thông số như sau :

    ```
    Option Base 1
    Dim matran(100,100) As Double
    ```

  - nhưng nếu đại đa số lần dùng ứng dụng này, ta chỉ giải các hệ phương trình có 2, 3,... ẩn số thì sẽ rất phí phạm bộ nhớ. Còn 1 lần chạy nào đó, nếu ta cần giải hệ 200 phương trình thì chương trình sẽ chạy sai.
- khai báo số lượng động tại thời điểm chạy. Cú pháp như sau :

  Dim *varname***()** [**As** [**New**] *type*]

  Ví dụ : Dim matran() As Double 'để trống số lượng

  ...

  n = Val(txtInput.Text)

  ReDim matran(n,n) 'phân phối phần tử cho ma trận

6.6 Lệnh khai báo Declare

- Các lệnh định nghĩa hằng, biến, kiểu, thủ tục cho phép ta sản sinh phần tử tương ứng trong phạm vi ngữ cảnh tương ứng (thủ tục, module, toàn cục).
- Ngoài ra Windows (và nhiều hãng, cá nhân khác) đã viết nhiều module tổng quát, mỗi module chứa nhiều thủ tục khác nhau, các thủ tục này giải quyết những vấn đề nào đó. Thí dụ ta có module các hàm lượng giác, module các hàm thống kê, module các hàm xử lý dữ liệu multimedia,...
- Windows dùng kỹ thuật liên kết động các module trên vào ứng dụng dùng chúng, mỗi module được cất trên 1 file *.dll (dynamic link library).
- VB cung cấp lệnh khai báo "Declare" để cho phép người lập trình khai báo chữ ký (signature, interface, prototype, header,...) của các thủ tục có sẵn trong các module *.dll để gọi nó trong ngữ cảnh của mình (module).

Lệnh khai báo Declare (tt)

Cú pháp 1 :

[**Public | Private**] **Declare Sub** *name* **Lib** "*libname*" [**Alias** "*aliasname*"] [([*arglist*])]

Cú pháp 2 :

[**Public | Private**] **Declare Function** *name* **Lib** "*libname*" [**Alias** "*aliasname*"] [([*arglist*])] [As *type*]

- Cú pháp 1 cho phép khai báo 1 subroutine với tên là *name* ở thư viện tên là *libname*, ta có thể gọi subroutine này bằng 1 tên khác là *aliasname* và truyền cho nó 1 danh sách đối số tương thích với *arglist*.
- Cú pháp 2 cho phép khai báo 1 function với tên là *name* ở thư viện tên là *libname*, ta có thể gọi function này bằng 1 tên khác là *aliasname* và truyền cho nó 1 danh sách đối số tương thích với *arglist*. Sau khi hoàn thành, function sẽ trả về 1 giá trị kết quả thuộc kiểu *type*.
- Chi tiết về sự khác biệt giữa subroutine và function sẽ được trình bày trong chương 9 và 10.

Thí dụ về các lệnh định nghĩa VB

Chúng ta đã trình bày qui trình thiết kế trực quan giao diện của trình MiniCalculator cho phép giả lập 1 máy tính tay đơn giản. Chương trình này chỉ có 1 form, trong form này chúng ta sẽ định nghĩa các hằng, biến cục bộ sau đây để phục vụ hoạt động cho ứng dụng :

```
Option Explicit
Const IDC_EQUAL = 0          ' định nghĩa các hằng gợi nhớ miêu tả toán tử
Const IDC_ADD = 1
Const IDC_SUB = 2
Const IDC_MUL = 3
Const IDC_DIV = 4

Private dblDispValue As Double          ' biến lưu giá trị đang hiển thị
Private dblOldValue As Double           ' biến lưu giá trị trước đó
Private dblMemValue As Double           ' biến lưu giá trị trong bộ nhớ
Private blnFpoint As Boolean            ' trạng thái nhập số nguyên/lẻ
Private bytPosDigit As Byte             ' vị trí lý số lẻ đang nhập
Private intPosNeg As Integer            ' trạng thái miêu tả giá trị âm/dương
Private bytOperationId As Byte          ' id của phép toán cần thực hiện
Private blnFAfterOp As Boolean          ' trạng thái nhập ký số đầu sau phép toán
```


MÔN TIN HỌC

Chương 7

BIỂU THỨC VB

7.1 Tổng quát về biểu thức VB
7.2 Các toán tử
7.3 Qui trình tính biểu thức
7.4 Quyền ưu tiên của các toán tử

7.1 Tổng quát về biểu thức VB

- Ta đã biết trong toán học công thức là phương tiện miêu tả 1 qui trình tính toán nào đó trên các số.
- Trong VB (hay ngôn ngữ lập trình khác), ta dùng biểu thức để miêu tả qui trình tính toán nào đó trên các dữ liệu ⇒ biểu thức cũng giống như công thức toán học, tuy nó tổng quát hơn (xử lý trên nhiều loại dữ liệu khác nhau) và phải tuân theo qui tắc cấu tạo khắt khe hơn công thức toán học.
- Để hiểu được biểu thức, ta cần hiểu được các thành phần của nó :
  - Các toán hạng : các biến, hằng dữ liệu,...
  - Các toán tử tham gia biểu thức : +,-,*,/,...
  - Qui tắc kết hợp toán tử và toán hạng để tạo biểu thức.
  - Qui trình mà máy dùng để tính trị của biểu thức.
  - Kiểu của biểu thức là kiểu của kết quả tính toán biểu thức.

Các biểu thức cơ bản

Biểu thức cơ bản là phần tử nhỏ nhất cấu thành biểu thức bất kỳ. Một trong các phần tử sau được gọi là biểu thức cơ bản :

- Biến,
- Hằng gợi nhớ,
- Giá trị dữ liệu cụ thể thuộc kiểu nào đó (nguyên, thực,..)
- Lời gọi hàm,
- 1 biểu thức được đóng trong 2 dấu ().

Qui trình tạo biểu thức là qui trình đệ qui : ta kết hợp từng toán tử với các toán hạng của nó, trong đó toán hạng hoặc là biểu thức cơ bản hoặc là biểu thức sẵn có (đã được xây dựng trước đó và nên đóng trong 2 dấu () để biến nó trở thành biểu thức cơ bản).

7.2 Các toán tử

Dựa theo số toán hạng tham gia, có 2 loại toán tử thường dùng nhất :

- **toán tử 1 ngôi** : chỉ cần 1 toán hạng. Ví dụ toán tử '-' để tính phần âm của 1 đại lượng.
- **toán tử 2 ngôi** : cần dùng 2 toán hạng. Ví dụ toán tử '*' để tính tích của 2 đại lượng.

VB thường dùng các ký tự đặc biệt để miêu tả toán tử. Ví dụ :

- **toán tử '+'** : cộng 2 đại lượng.
- **toán tử '-'** : trừ đại lượng 2 ra khỏi đại lượng 1.
- **toán tử '*'** : nhân 2 đại lượng.
- **toán tử '/'** : chia đại lượng 1 cho đại lượng 2...

Trong vài trường hợp, VB dùng cùng 1 ký tự đặc biệt để miêu tả nhiều toán tử khác nhau. Trong trường hợp này, ngữ cảnh sẽ được dùng để giải quyết nhằm lẫn.

Ngữ cảnh thường là kiểu của các toán hạng tham gia hoặc do thiếu toán hạng thì toán tử được hiểu là toán tử 1 ngôi.

Các toán tử (tt)

Dựa theo độ ưu tiên của các toán tử trong qui trình tính toán biểu thức, có 3 loại toán tử :

- **toán tử số học** : có độ ưu tiên cao nhất trong qui trình tính toán biểu thức.
- **toán tử so sánh** : có độ ưu tiên kế tiếp.
- **toán tử luận lý và bitwise** : có độ ưu tiên thấp nhất.

Trong các slide sau, chúng ta sẽ trình bày chi tiết các toán tử VB thuộc từng loại trên.

Các toán tử số học

Tùy thuộc kiểu của các toán hạng tham gia mà ta được phép dùng những toán tử nào trên chúng ⇒ số lượng toán tử có giá trị trên từng kiểu dữ liệu là khác nhau ⇒ phải học và nhớ từ từ.

Dữ liệu số là loại dữ liệu thường được xử lý nhất trong các ứng dụng (may mắn cho chúng ta vì ta đã quen với toán học).

Các toán tử trên dữ liệu số là :

- **toán tử '&'** : nối kết 2 chuỗi thành 1 chuỗi.
- **toán tử '+'** : cộng 2 đại lượng.
- **toán tử '-'** : trừ đại lượng 2 ra khỏi đại lượng 1.
- **toán tử '*'** : nhân 2 đại lượng.
- **toán tử '/'** : chia đại lượng 1 cho đại lượng 2.
- **toán tử '\'** : chia nguyên.
- **toán tử Mod** : lấy phần dư của phép chia nguyên.
- **toán tử '^'** : lũy thừa.

Toán tử '&' để nối kết 2 chuỗi

Cú pháp :

expr1 **&** *expr2* (→ kết quả)

- nối kết 2 toán hạng kiểu chuỗi thành 1 chuỗi mới, nếu 1 trong 2 toán hạng thuộc kiểu số thì nó sẽ được đổi thành dạng chuỗi trước khi thực hiện nối kết.

Ví dụ :

```
Dim MyStr As String
MyStr = "Hello" & " World"      ' kết quả là "Hello World".
MyStr = "Check " & 123 & " Check"   ' kq là "Check 123 Check".
```

- lưu ý nên có ký tự trống trong các chuỗi con sao cho nối kết chuỗi kết quả dễ đọc.

Toán tử '+' trên dữ liệu số

Cú pháp :

expr1 **+** *expr2* (→ kết quả) hoặc **+** *expr1*

Nếu cả 2 toán hạng đều là số thì kiểu kết quả là kiểu chính xác nhất của phép + theo thứ tự sau : Byte, Integer, Long, Single, Double, Currency, Decimal với các ngoại lệ sau :

Nếu	thì kết quả là :
1 toán hạng Single,1 toán hạng Long	Double
kết quả kiểu Variant chứa giá trị Single, Long, Date và bị tràn	Variant chứa Double
kết quả kiểu Variant chứa giá trị Byte và bị tràn	Variant chứa Integer
kết quả kiểu Variant chứa giá trị Integer và bị tràn	Variant chứa Long
1 toán hạng Date,1 toán hạng kiểu khác	Date

Toán tử '+' trên dữ liệu số (tt)

Nếu kiểu của cả 2 toán hạng đều là Variant thì việc xác định ngữ nghĩa phép + và kiểu kết quả sẽ theo qui luật của bảng sau :

Nếu	thì :
cả 2 toán hạng là Variant chứa số	Cộng
cả 2 toán hạng là Variant chứa chuỗi	Nối kết 2 chuỗi
1 là Variant chứa số, 1 là Variant chứa chuỗi	Cộng

Toán tử '+' trên dữ liệu số (tt)

Nếu ít nhất 1 toán hạng không phải Variant thì việc xác định ngữ nghĩa phép + và kiểu kết quả sẽ theo qui luật của bảng sau :

Nếu	thì :
cả 2 toán hạng là dữ liệu số	Cộng
cả 2 toán hạng là chuỗi	Nối kết 2 chuỗi
1 là số, 1 là Variant giá trị khác Null	Cộng
1 là chuỗi, 1 là Variant giá trị khác Null	Nối kết 2 chuỗi
1 biểu thức là Variant chứa Empty	kết quả là toán hạng còn lại
1 là số và 1 là chuỗi	A Type mismatch error
1 trong 2 toán hạng là Null	kết quả là Null

Toán tử '-' trên dữ liệu số

Cú pháp :

expr1 **-** *expr2* (→ kết quả) hoặc **-** *expr1*

Kiểu kết quả thường là kiểu chính xác nhất của phép - theo thứ tự sau : Byte, Integer, Long, Single, Double, Currency, Decimal với các ngoại lệ sau :

Nếu	thì kết quả là :
1 toán hạng Single,1 toán hạng Long	Double
kết quả kiểu Variant chứa giá trị Single, Long, Date và bị tràn	Variant chứa Double
kết quả kiểu Variant chứa giá trị Integer và bị tràn	Variant chứa Long
1 toán hạng Date,1 toán hạng kiểu khác	Date
cả 2 toáng hạng Date	Double

Toán tử '*' trên dữ liệu số

Cú pháp :

expr1 * *expr2* (→ kết quả)

Kiểu kết quả thường là kiểu chính xác nhất của phép * theo thứ tự sau : Byte, Integer, Long, Single, Double, Currency, Decimal với các ngoại lệ sau :

Nếu	thì kết quả là :
1 toán hạng Single,1 toán hạng Long	Double
kết quả kiểu Variant chứa giá trị Single, Long, Date và bị tràn	Variant chứa Double
kết quả kiểu Variant chứa giá trị Byte và bị tràn	Variant chứa Integer
kết quả kiểu Variant chứa giá trị Integer và bị tràn	Variant chứa Long

Toán tử '/' trên dữ liệu số

Cú pháp :

expr1 / *expr2* (→ kết quả)

Kiểu kết quả thường là kiểu Double hay Variant chứa Double với các ngoại lệ sau :

Nếu	thì kết quả là :
cả 2 toán hạng là Byte, Integer,Single	Single, nếu tràn thì báo sai
cả 2 toán hạng là variant chứa trị Byte, Integer, Single	Variant chứa Single, nếu tràn thì đổi thành Variant chứa Double
1 toán hạng Decimal	Decimal

Toán tử '\' và Mod trên dữ liệu số

Cú pháp :

expr1 **** *expr2* (→ kết quả)

- Đây là phép chia nguyên, 2 toán hạng được đổi về dạng nguyên (được làm tròn) trước khi thực hiện phép chia.
- Kiểu kết quả hoặc là Byte, Integer, Long hoặc là Variant chứa trị Byte, Integer, Long.

Ví dụ : 19 \ 6.7 → kết quả là 2

Cú pháp :

expr1 **Mod** *expr2* (→ kết quả)

- Đây là phép lấy phần dư của phép chia nguyên, 2 toán hạng được đổi về dạng nguyên (được làm tròn) trước khi thực hiện phép chia.
- Kiểu kết quả hoặc là Byte, Integer, Long hoặc là Variant chứa trị Byte, Integer, Long.

Ví dụ : 19 Mod 6.7 → kết quả là 5

Toán tử '^' trên dữ liệu số

Cú pháp :

number **^** *exponent* (→ kết quả)

- Đây là phép lũy thừa, 2 toán hạng thuộc kiểu số (Byte, Integer, Long, Single, Double,...) với hạn chế là nếu phần mũ là số nguyên thì phần cơ số (number) mới được phép âm.
- Kiểu kết quả hoặc là Double hoặc là Variant chứa trị Double.

Ví dụ : (-5) ^ 3 → kết quả là -125.0

3 ^ 3 ^ 3 → kết quả là 19683.0

3.2 ^ 2.7 → kết quả là 23.115587799

Các toán tử so sánh dữ liệu

Cú pháp :

expr1 op expr2 (→ kết quả)

- 2 toán hạng thường là kiểu số hay chuỗi. Kết quả luôn là kiểu luận lý (nhận 1 trong 2 trị True, False).
- op là 1 trong các toán tử so sánh sau :
  - < : phép toán nhỏ hơn
  - <= : phép toán nhỏ hơn hoặc bằng
  - > : phép toán lớn hơn
  - >= : phép toán lớn hơn hoặc bằng
  - = : phép toán so sánh bằng
  - <> : phép toán khác nhau (không bằng)

Ngoài các toán tử so sánh thông thường trên, VB còn cung cấp 2 toán tử so sánh đặc biệt sau (với ngữ nghĩa đặc biệt sẽ được trình bày trong các slide sau) :

expr1 **Like** *expr2* (→ kết quả)

expr1 **Is** *expr2* (→ kết quả)

Toán tử Like

Cú pháp :

string **Like** *pattern* (→ kết quả)

- xác định xem chuỗi cụ thể *string* có thuộc về *pattern* không. Nếu thuộc về thì cho kết quả True, nếu không thuộc về thì cho kết quả False.

Ví dụ :

```
MyCheck = "aBBBa" Like "a*a"          ' Returns True.
MyCheck = "F" Like "[A-Z]"            ' Returns True.
MyCheck = "F" Like "[!A-Z]"           ' Returns False.
MyCheck = "a2a" Like "a#a"            ' Returns True.
MyCheck = "aM5b" Like "a[L-P]#[!c-e]" ' Returns True.
MyCheck = "BAT123khg" Like "B?T*"     ' Returns True.
MyCheck = "CAT123khg" Like "B?T*"     ' Returns False.
```

Toán tử Like (tt)

Hành vi của toán tử Like phụ thuộc vào 1 trong 2 chế độ do phát biểu "Option Compare" qui định :

- **Option Compare Binary** ' default
- **Option Compare Text**

Trong chế độ so sánh Binary, VB dựa vào thứ tự sắp xếp các ký tự trên cơ sở mã nhị phân của các ký tự. Trong bảng mã ISO8859-1, ta có :

A < B < E < Z < a < b < e < z < À < Ê < Ø < à < ê < ø

Trong chế độ so sánh Text, VB dựa vào thứ tự sắp xếp các ký tự trên cơ sở ngữ nghĩa ký tự và thông tin "locale" của Windows (do đó không phân biệt chữ thường và hoa) :

(A=a) < (À=à) < (B=b) < (E=e) < (Ê=ê) < (Z=z) < (Ø=ø)

Thông tin về chế độ so sánh cũng được áp dụng cho các toán tử so sánh thông thường trên các chuỗi.

Toán tử Like (tt)

Toán hạng *string* là chuỗi ký tự cụ thể, còn toán hạng *pattern* là chuỗi chứa các ký tự cụ thể và/hoặc các ký tự đặc biệt có ý nghĩa theo bảng sau :

Ký tự trong pattern	Tương ứng với :
?	Bất kỳ 1 ký tự nào
*	bất kỳ chuỗi ký tự nào (dài từ 0 ký tự trở lên)
#	Bất kỳ ký số thập phân nào (0–9).
[!*charlist*]	Bất kỳ ký tự không có trong *charlist*.
[*charlist*]	Bất kỳ ký tự có trong *charlist*.

- dùng '-' để miêu tả 1 phạm vi xác định bởi 2 cận dưới và trên.
- dùng cú pháp [c] để miêu tả các ký tự đặc biệt.

Toán tử Is

Cú pháp :

ObjVar1 **Is** *ObjVar1* (→ kết quả)

- xác định xem 2 biến ObjVar1 và ObjVar2 có chứa cùng tham khảo đến 1 đối tượng duy nhất không. Nếu đúng vậy thì kết quả của biểu thức là True, nếu không trị biểu thức là False.

Ví dụ :

```
Dim MyObject, YourObject, ThisObject, ThatObject
Dim MyCheck As Boolean
Set YourObject = New Clipboard     ' tạo object và gán tham khảo.
Set ThisObject = YourObject
Set ThatObject = New Clipboard
MyCheck = YourObject Is ThisObject          ' kết quả True.
MyCheck = ThatObject Is ThisObject          ' kết quả False.
```


Các toán tử luận lý

Các toán tử luận lý cho phép thực hiện 1 hành vi luận lý trên 1 hay 2 toán hạng thuộc kiểu luận lý để cho kết quả là 1 giá trị luận lý.

Các toán tử luận lý là :

- **toán tử And** : phép toán 'và'.
- **toán tử Or** : phép toán 'hoặc'.
- **toán tử Xor** : phép toán loại trừ.
- **toán tử Not** : phép toán đảo.
- **toán tử Eqv** : phép toán tương đương.
- **toán tử Imp** : phép toán kéo theo.

Nếu cả 2 toán hạng đều là số thì các phép toán trên sẽ thực hiện hành vi của chúng trên từng cặp bit tương ứng của 2 toán hạng (sẽ giải thích cụ thể sau).

Toán tử luận lý And

Cú pháp :

expr1 **And** *expr2* (→ kết quả)

- kết quả được xác định theo bảng sau :

Ghi chú :

- o kết quả chỉ True khi cả 2 toán hạng là True.
- o Kết quả là False nếu có 1 toán hạng là False.

expr1	expr2	kết quả
True	True	True
True	False	False
True	Null	Null
False	True	False
False	False	False
False	Null	False
Null	True	Null
Null	False	False
Null	Null	Null

Toán tử bitwise And

Cú pháp :

expr1 **And** *expr2* (→ kết quả)

- 2 toán hạng thuộc kiểu số và toán tử And thực hiện hành vi trên từng cặp bit tương ứng của 2 toán hạng, kết quả trên từng cặp bit được xác định theo bảng sau :

biti of expr1	biti of expr2	kết quả
1	1	1
1	0	0
0	1	0
0	0	0

Toán tử luận lý Or

Cú pháp :

expr1 **Or** *expr2* (→ kết quả)

- kết quả được xác định theo bảng sau :

Ghi chú :

- o kết quả là True nếu có 1 toán hạng là True.
- o Kết quả chỉ False khi cả 2 toán hạng đều là False.

expr1	expr2	kết quả
True	True	True
True	False	True
True	Null	True
False	True	True
False	False	False
False	Null	Null
Null	True	True
Null	False	Null
Null	Null	Null

Toán tử bitwise Or

Cú pháp :

expr1 **Or** *expr2* (→ kết quả)

- 2 toán hạng thuộc kiểu số và toán tử Or thực hiện hành vi trên từng cặp bit tương ứng của 2 toán hạng, kết quả trên từng cặp bit được xác định theo bảng sau :

biti of expr1	biti of expr2	kết quả
1	1	1
1	0	1
0	1	1
0	0	0

Toán tử luận lý Xor

Cú pháp :

expr1 **Xor** *expr2* (→ kết quả)

- kết quả được xác định theo bảng sau :

Ghi chú :

- Nếu có 1 toán hạng là Null thì kết quả là Null.
- Kết quả là True nếu 2 toán hạng khác nhau và khác Null.
- Kết quả là False nếu 2 toán hạng giống nhau và khác Null.

expr1	expr2	kết quả
True	True	False
True	False	True
True	Null	Null
False	True	True
False	False	False
False	Null	Null
Null	True	Null
Null	False	Null
Null	Null	Null

Toán tử bitwise Xor

Cú pháp :

expr1 **Xor** *expr2* (→ kết quả)

- 2 toán hạng thuộc kiểu số và toán tử Xor thực hiện hành vi trên từng cặp bit tương ứng của 2 toán hạng, kết quả trên từng cặp bit được xác định theo bảng sau :

biti of expr1	biti of expr2	kết quả
1	1	0
1	0	1
0	1	1
0	0	0

Toán tử luận lý Eqv

Cú pháp :

expr1 **Eqv** *expr2* (→ kết quả)

- kết quả được xác định theo bảng sau :

expr1	expr2	kết quả
True	True	True
True	False	False
True	Null	Null
False	True	False
False	False	True
False	Null	Null
Null	True	Null
Null	False	Null
Null	Null	Null

Ghi chú :

- Nếu có 1 toán hạng là Null thì kết quả là Null.
- Kết quả là True nếu 2 toán hạng giống nhau và khác Null.
- Kết quả là False nếu 2 toán hạng khác nhau và khác Null.

Toán tử bitwise Eqv

Cú pháp :

expr1 **Eqv** *expr2* (→ kết quả)

- 2 toán hạng thuộc kiểu số và toán tử Eqv thực hiện hành vi trên từng cặp bit tương ứng của 2 toán hạng, kết quả trên từng cặp bit được xác định theo bảng sau :

biti of expr1	biti of expr2	kết quả
1	1	1
1	0	0
0	1	0
0	0	1

Toán tử luận lý Imp

Cú pháp :

expr1 **Imp** *expr2* (→ kết quả)

- kết quả được xác định theo bảng sau :

Ghi chú :

- o Nếu toán hạng 1 là False thì kết quả là True.
- o Kết quả là True nếu 2 toán hạng đều là True.

expr1	expr2	kết quả
True	True	True
True	False	False
True	Null	Null
False	True	True
False	False	True
False	Null	True
Null	True	True
Null	False	Null
Null	Null	Null

Toán tử bitwise Imp

Cú pháp :

expr1 **Imp** *expr2* (→ kết quả)

- 2 toán hạng thuộc kiểu số và toán tử Imp thực hiện hành vi trên từng cặp bit tương ứng của 2 toán hạng, kết quả trên từng cặp bit được xác định theo bảng sau :

biti of expr1	biti of expr2	kết quả
1	1	1
1	0	0
0	1	1
0	0	1

Toán tử luận lý và bitwise Not

Cú pháp :

Not *expr* (→ kết quả)

- kết quả của biểu thức theo bảng sau :

expr	kết quả
True	False
False	True
Null	Null

biti of expr	kết quả
1	0
0	1

Cú pháp :

Not *expr* (→ kết quả)

- toán hạng thuộc kiểu số và toán tử Not thực hiện hành vi trên từng bit tương ứng của toán hạng, kết quả trên từng bit được xác định theo bảng trên :

7.3 Qui trình tính biểu thức

Nếu biểu thức được xây dựng chỉ trên các biểu thức cơ bản thì qui trình tính biểu thức chính là qui trình xây dựng biểu thức đó.

Nếu biểu thức được xây dựng trên các biểu thức con bất kỳ thì qui trình tính toán như sau : tính từ trái sang phải, mỗi lần gặp 1 toán tử (ký hiệu là CurrentOp) thì phải nhìn trước toán tử đi ngay sau nó (SussesorOp), so sánh độ ưu tiên của 2 toán tử và ra quyết định như sau :

- nếu không có SussesorOp thì tính ngay toán tử CurrentOp (trên 1 hay 2 toán hạng của nó).
- nếu toán tử CurrentOp có độ ưu tiên cao hơn hay bằng toán tử SussesorOp thì tính ngay toán tử CurrentOp (trên 1 hay 2 toán hạng của nó).
- nếu toán tử CurrentOp có độ ưu tiên thấp hơn SussesorOp thì cố gắng thực hiện toán tử SussesorOp trước. Việc cố gắng này cũng có thể bị tạm hoãn nếu toán tử đi sau toán tử SussesorOp có độ ưu tiên cao hơn SussesorOp,...
- Khi toán tử SussesorOp được thực hiện xong thì toán tử ngay sau SussesorOp trở thành toán tử đi ngay sau CurrentOp ⇒ việc kiểm tra xem CurrentOp có được thực hiện không sẽ được lặp lại.

7.4 Thứ tự ưu tiên cụ thể của các toán tử

Các toán tử số học có độ ưu tiên cao nhất, rồi tới các toán tử so sánh và sau cùng là các toán tử luận lý :

- Giữa các toán tử số học, quyền ưu tiên từ cao xuống thấp theo tứ tự từ trên xuống trong bảng sau.
- Các toán tử so sánh có cùng thứ tự ưu tiên.
- Giữa các toán tử luận lý, quyền ưu tiên từ cao xuống thấp theo tứ tự từ trên xuống trong bảng sau.

Arithmetic	Comparison	Logical
1.Exponentiation (^)	8.Equality (=)	9. Not
2.Negation (–)	8.Inequality (<>)	10.And
3.Multiplication and division (*, /)	8.Less than (<)	11.Or
4.Integer division (\)	8.Greater than (>)	12.Xor
5.Modulus arithmetic (Mod)	8.Less than or equal to (<=)	13.Eqv
6.Addition and subtraction (+, –)	8.Greater than or equal to (>=)	14.Imp
7.String concatenation (&)	8.Like, Is	

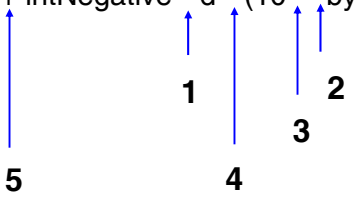
Thí dụ về qui trình tính biểu thức

Thí dụ sau là biểu thức tính giá trị mới của phần tử Display trong chương trình MiniCalculator trong trường hợp người dùng mới nhập thêm 1 ký số ở phần lẻ, trong đó :

- dblDispValue là biến chứa trị của Display.
- intNegative là biến miêu tả dấu của trị Display (1 : dương, -1 : âm).
- d là biến chứa ký số mới nhập.
- bytPosDigit là biến miêu tả vị trí ký số vừa nhập ở bên phải dấu '.'.

```
dblDispValue = dblDispValue + intNegative * d * (10 ^ -bytPosDigit)
```

Ghi chú : theo thứ tự, toán tử lũy thừa được tính trước toán tử -, nhưng ở đây để tính được luỹ thừa, ta buộc phải xác định được toán hạng đi sau nó và như vậy toán tử - phải được tính trước trong trường hợp này.



MÔN TIN HỌC

Chương 8

CÁC LỆNH THỰC THI VB

8.1 Tổng quát về ngôn ngữ VB
8.2 Các lệnh gán.
8.3 Các lệnh kiểm tra điều kiện & rẽ nhánh
8.4 Các lệnh lặp
8.5 Vấn đề lồng nhau giữa các lệnh
8.6 Thoát đột ngột khỏi khỏi cấp điều khiển
8.7 Lệnh gọi hàm/thủ tục

8.1 Tổng quát về các lệnh thực thi VB

- Ta đã biết giải thuật để giải quyết 1 vấn đề nào đó là trình tự các công việc nhỏ hơn, nếu ta thực hiện đúng trình tự các công việc nhỏ hơn này thì sẽ giải quyết được vấn đề lớn.
- VB (hay ngôn ngữ lập trình khác) cung cấp 1 tập các lệnh thực thi, mỗi lệnh thực thi được dùng để miêu tả 1 công việc nhỏ trong 1 giải thuật với ý tưởng chung như sau :
  - Nếu tồn tại lệnh thực thi miêu tả được công việc nhỏ của giải thuật thì ta dùng lệnh thực thi này.
  - Nếu công việc nhỏ vẫn còn quá phức tạp và không có lệnh thực thi nào miêu tả được thì ta dùng lệnh gọi thủ tục (Function, Sub, Property) trong đó thủ tục là trình tự các lệnh thực hiện công việc nhỏ này...
- Hầu hết các lệnh thực thi có chứa biểu thức và dùng kết quả của biểu thức này để quyết định công việc kế tiếp cần được thực hiện $\Rightarrow$ ta thường gọi các lệnh thực thi là các cấu trúc điều khiển.

Tổng quát về các lệnh thực thi VB (tt)

- Để dễ học, dễ nhớ và dễ dùng, VB (cũng như các ngôn ngữ khác) chỉ cung cấp 1 số lượng rất nhỏ các lệnh thực thi :
  - **Nhóm lệnh không điều khiển** :
    - Lệnh gán dữ liệu vào 1 biến.
    - Lệnh gán tham khảo đến đối tượng vào 1 biến tham khảo.
  - **Nhóm lệnh tạo quyết định :**
    - Lệnh kiểm tra điều kiện luận lý **If ... Then ... Else**
    - Lệnh kiểm tra điều kiện số học **Select Case**
  - **Nhóm lệnh lặp :**
    - Lệnh lặp **Do ... Loop**
    - Lệnh lặp **For ... Next**
    - Lệnh lặp **For Each ... Next**
  - **Nhóm lệnh gọi thủ tục :**
    - Lệnh gọi thủ tục
    - Lệnh thoát khỏi cấu trúc điều khiển **Exit**

8.2 Lệnh gán dữ liệu

- Lệnh được dùng nhiều nhất trong 1 chương trình là lệnh gán giá trị dữ liệu vào 1 vùng nhớ để lưu trữ lại dữ liệu này hầu sử dụng lại nó sau đó. Chúng ta đã thấy lệnh này nhiều lần trong các chương trước, bây giờ chúng ta nói rõ hơn về nó.

Cú pháp :

lvar = *expr*

- biểu thức bên phải sẽ được tính để tạo ra kết quả (1 giá trị cụ thể thuộc 1 kiểu cụ thể), giá trị này sẽ được gán vào ô nhớ do *lvar* qui định. Trước khi gán, VB sẽ kiểm tra kiểu của 2 phần tử (qui tắc kiểm tra sẽ được trình bày sau).
- lvar thường là 1 biến dữ liệu cơ bản, nhưng có thể đệ qui theo qui tắc :
  - nếu lvar là biến dãy thì 1 phần tử dãy có thể là lvar.
  - nếu lvar là biến dữ liệu người dùng thì 1 field của nó có thể là lvar.
  - nếu lvar là biến đối tượng thì 1 thuộc tính của đối tượng có thể là lvar.

Ví dụ :

```
dblDispValue = dblDispValue + intNegative * d * (10 ^ -bytPosDigit)
```

Lệnh gán tham khảo đến đối tượng

- Như đã được trình bày trong chương 5, biến đối tượng (có kiểu là Object hay tên class module nào đó) chỉ chứa tham khảo đến đối tượng chứ không chứa trực tiếp đối tượng. Khi mới định nghĩa, những biến này chưa tham khảo đến đối tượng cụ thể nào, do đó trước khi dùng chúng, ta phải gán tham khảo của đối tượng cụ thể vào biến.

Cú pháp :

Set *lvar* **=** *expr*

- biểu thức bên phải sẽ được tính để tạo ra kết quả là 1 tham khảo đến đối tượng, tham khảo này sẽ được gán vào ô nhớ do *lvar* qui định. Trước khi gán, VB sẽ kiểm tra kiểu của 2 phần tử (qui tắc kiểm tra sẽ được trình bày sau).
- lvar thường là 1 biến đối tượng cơ bản, nhưng có thể đệ qui theo qui tắc :
  - nếu lvar là biến dãy thì 1 phần tử dãy có thể là lvar.
  - nếu lvar là biến dữ liệu người dùng thì 1 field của nó có thể là lvar.
  - nếu lvar là biến đối tượng thì 1 thuộc tính của đối tượng có thể là lvar.

Ví dụ :

```
Set objClipbd = New Clipboard
```


8.3 Lệnh kiểm tra điều kiện luận lý IF

- Cho phép dựa vào kết quả luận lý (tính được từ 1 biểu thức luận lý) để quyết định thi hành 1 trong 2 nhánh lệnh. Sau khi thực hiện 1 trong 2 nhánh lệnh, chương trình sẽ tiếp tục thi hành lệnh ngay sau lệnh IF. Có nhiều cú pháp khác nhau :

Cú pháp 1a :

If *condition* **Then** *Statement1* **[Else** *Statement2* **]**

- *condition* là 1 biểu thức luận lý miêu tả điều kiện cần kiểm tra, nó có kết quả True/False.
- *Statement1, Statement2* là lệnh thực thi VB bất kỳ.
- nếu kết quả là True thì thi hành *Statement1.*
- nếu kết quả là False và có dùng Else thì thi hành *Statement2*.

Ví dụ :

```
If blnThaybenh Then MsgBox("Thầy bệnh. Sinh viên về nghỉ")
```

Lệnh kiểm tra điều kiện luận lý IF (tt)

Cú pháp 2 :

If *condition* **Then**
[Statement]+
End If

- *condition* là 1 biểu thức luận lý miêu tả điều kiện cần kiểm tra, nó có kết quả True/False.
- *[Statement]+* là danh sách các lệnh thực thi VB bất kỳ.
- nếu kết quả là True thì thi hành các lệnh *[Statement]+*, nếu kết quả là False thi thôi.

Ví dụ :

```
If del >=0 Then
   x1 = (-b-sqr(del))/(2*a)
   x2 = (-b+sqr(del))/(2*a)
   MsgBox("x1= " & x1 & " ,x2= " & x2)
End If
```


Lệnh kiểm tra điều kiện luận lý IF (tt)

Cú pháp 3 :

If *condition* **Then**
[Statement1]+
Else
[Statement2]+
End If

- *condition* là 1 biểu thức luận lý miêu tả điều kiện cần kiểm tra, nó có kết quả True/False.
- *[Statement1]+, [Statement2]+* là danh sách các lệnh thực thi VB bất kỳ.
- nếu kết quả là True thì thi hành các lệnh *[Statement1]+*, nếu kết quả là False thì thi hành các lệnh *[Statement2]+*.

Lệnh kiểm tra điều kiện luận lý IF (tt)

Ví dụ :

```
If del >=0 Then
    x1 = (-b-sqr(del))/(2*a)
    x2 = (-b+sqr(del))/(2*a)
    MsgBox("x1= " & x1 & " ,x2= " & x2)
Else
    MsgBox("Phương trình vô nghiệm")
End If
```

Ví dụ : hiệu chỉnh trị phần tử Display khi người dùng nhập thêm ký số d

```
If (blnFpoint) Then     ' phần lẻ
   bytPosDigit = bytPosDigit + 1
   dblDispValue = dblDispValue + intPosNeg * d * (10 ^ -bytPosDigit)
Else          ' phần nguyên
   dblDispValue = dblDispValue * 10 + intPosNeg * d
End If
```


Lệnh kiểm tra điều kiện số học Select

Cú pháp :

Select Case *condition*
Case *expr1*
[Statement1]+
Case *expr2*
[Statement2]+
...
Case Else
[Statementn]+
End Select

- *condition* là 1 biểu thức số học miêu tả điều kiện cần kiểm tra, nó có giá trị số.
- *[Statement1]+, [Statement2]+* là danh sách các lệnh thực thi VB bất kỳ.
- tùy giá trị của điều kiện trùng với nhánh Case nào mà các lệnh VB trong nhánh đó được thi hành, sau đó VB sẽ thi hành lệnh ngay sau lệnh Select.

Lệnh kiểm tra điều kiện số học Select (tt)

Thí dụ sau là lệnh Select phục vụ việc thực hiện 1 nút lệnh trong trình MiniCalculator mà ta sẽ thực hành :

```
Select Case bytOperationId
   Case IDC_ADD      ' phép cộng
      dblDispValue = dblOldValue + dblDispValue
      txtDisplay.Text = Str(dblDispValue)
   Case IDC_SUB      ' phép trừ
      dblDispValue = dblOldValue - dblDispValue
      txtDisplay.Text = Str(dblDispValue)
   Case IDC_MUL      ' phép nhân
      dblDispValue = dblOldValue * dblDispValue
      txtDisplay.Text = Str(dblDispValue)
   Case IDC_DIV      ' phép chia
      dblDispValue = dblOldValue / dblDispValue
      txtDisplay.Text = Str(dblDispValue)
End Select
```


Lệnh kiểm tra điều kiện số học On...GoSub

VB còn cung cấp 1 lệnh khác để kiểm tra điều kiện số học, nhưng yếu hơn lệnh Select, đó là lệnh On...GoSub (thực ra đây là lệnh của ngôn ngữ Basic nguyên thủy, Microsoft thấy chưa trong sáng nên mới cung cấp thêm lệnh Select). Cú pháp như sau :

On *condition* **GoSub** *label1, label2, label3,...*

- *condition* là 1 biểu thức số học và nên có giá trị từ 1 tới n, trong đó n là số lượng nhãn lệnh được liệt kê sau từ khóa GoSub.
- nếu giá trị của *condition* là i thì máy sẽ gọi 'subroutine' bắt đầu từ lệnh có nhãn là labeli.
- 'subroutine' là 1 danh sách gồm nhiều lệnh để thực hiện 1 công việc nào đó (có thể lớn, có thể nhỏ) với đặc điểm lệnh cuối trong danh sách là lệnh Return để trả điều khiển về lệnh gọi GoSub.
- Để xác định được lệnh cần nhảy đến, VB cho phép dùng 1 nhãn gợi nhớ (danh hiệu) hay 1 nhãn số (số) kết hợp với lệnh cần tham khảo (đừng lạm dụng tính chất này để đặt nhãn cho mọi lệnh trong chương trình).

Lệnh kiểm tra điều kiện số học On...GoSub (tt)

Đoạn code sau có công dụng như slide thí dụ dùng lệnh Select case (nếu IDC_ADD = 1, IDC_SUB = 2, IDC_MUL = 3, IDC_DIV = 4) :

```
    On bytOperationId GoSub LblAdd, LblSub, LblMul, LblDiv
    ...
LblAdd:
    dblDispValue = dblOldValue + dblDispValue
    txtDisplay.Text = Str(dblDispValue)
    return          ' trả điều khiển về lệnh GoSub
LblSub:
    dblDispValue = dblOldValue - dblDispValue
    txtDisplay.Text = Str(dblDispValue)
    return          ' trả điều khiển về lệnh GoSub
LblMul:
    dblDispValue = dblOldValue * dblDispValue
    txtDisplay.Text = Str(dblDispValue)
    return          ' trả điều khiển về lệnh GoSub
LblDiv:
    dblDispValue = dblOldValue / dblDispValue
    txtDisplay.Text = Str(dblDispValue)
    return          ' trả điều khiển về lệnh GoSub
```


Lệnh kiểm tra điều kiện số học On...GoTo

1 biến thể khác của On...GoSub là On...Goto, ở đây điều khiển sẽ không trả lại lệnh On...Goto nữa. Đoạn code sau có công dụng như slide trước :

```
    On bytOperationId GoTo LblAdd, LblSub, LblMul, LblDiv
Continue:
    ...
LblAdd:
    dblDispValue = dblOldValue + dblDispValue
    txtDisplay.Text = Str(dblDispValue)
    Goto Continue       ' nhảy không điều kiện về nhãn Continue
LblSub:
    dblDispValue = dblOldValue - dblDispValue
    txtDisplay.Text = Str(dblDispValue)
    Goto Continue
LblMul:
    dblDispValue = dblOldValue * dblDispValue
    txtDisplay.Text = Str(dblDispValue)
    Goto Continue
LblDiv:
    dblDispValue = dblOldValue / dblDispValue
    txtDisplay.Text = Str(dblDispValue)
    Goto Continue
```

8.4 Lệnh lặp Do...Loop

Cú pháp 1 :

Do While *condition*
[Statement]+
Loop

- *condition* là 1 biểu thức luận lý miêu tả điều kiện cần kiểm tra, nó có kết quả True/False.
- *[Statement]+* là danh sách các lệnh thực thi VB bất kỳ.
- tính giá trị *condition*, nếu kết quả là True thì thi hành các lệnh *[Statement]+*, rồi lặp lại qui trình trên... đến lúc *condition* có giá trị False thì ngừng vòng lặp ⇒ thích hợp cho việc lặp từ 0 tới n lần.

Ví dụ : tính 10!

```
giaithua = 1
i = 1
Do While i <=10
    giaithua = giaithua * i
    i = i+1
Loop
```


Lệnh lặp While...Wend

- VB còn cung cấp 1 lệnh khác có chức năng giống như lệnh Do While ... Loop, cú pháp của nó như sau :

While *condition*
[Statement]+
Wend

- *condition* là 1 biểu thức luận lý miêu tả điều kiện cần kiểm tra, nó có kết quả True/False.
- *[Statement]+* là danh sách các lệnh thực thi VB bất kỳ.
- tính giá trị *condition*, nếu kết quả là True thì thi hành các lệnh *[Statement]+*, rồi lặp lại qui trình trên... đến lúc *condition* có giá trị False thì ngừng vòng lặp ⇒ thích hợp cho việc lặp từ 0 tới n lần.

Ví dụ : tính 10!

```
giaithua = 1
i = 1
While i <=10
    giaithua = giaithua * i
    i = i+1
Wend
```

Lệnh lặp Do...Loop (tt)

Cú pháp 2 :

Do

[Statement]+

Loop While *condition*

- *condition* là 1 biểu thức luận lý miêu tả điều kiện cần kiểm tra, nó có kết quả True/False.
- *[Statement]+* là danh sách các lệnh thực thi VB bất kỳ.
- thi hành các lệnh *[Statement]+* rồi kiểm tra *condition*, nếu có giá trị True thì lặp lại qui trình trên đến lúc nó có giá trị False thì ngừng vòng lặp ⇒ thích hợp cho việc lặp từ 1 tới n lần.

Ví dụ : tính 10!

```
giaithua = 1
i = 1
Do
    giaithua = giaithua * i
    i = i+1
Loop While i <=10
```


Lệnh lặp Do...Loop (tt)

Cú pháp 3 :

Do Until *condition*

[Statement]+

Loop

Cú pháp 4 :

Do

[Statement]+

Loop Until *condition*

- *condition* là 1 biểu thức luận lý miêu tả điều kiện cần kiểm tra, nó có kết quả True/False.
- *[Statement]+* là danh sách các lệnh thực thi VB bất kỳ.
- giống với cú pháp 1 và 2 nhưng thay vì điều kiện thực hiện vòng lặp là True thì bây giờ ngược lại là False mới thi hành vòng lặp.

Lệnh lặp For...Next

Cú pháp 1 :

For *counter* = *start* **To** *end* [**Step** *increment*]
[Statement]+
Next ***[****counter]*

- *counter* là biến điều khiển số lần lặp, *start* là biểu thức qui định giá trị đầu của *counter*, *end* qui định giá trị cuối, *increment* miêu tả bước tăng (âm là giảm, default là 1).
- *[Statement]+* là danh sách các lệnh thực thi VB bất kỳ.
- thi hành các lệnh *[Statement]+* với số lần được qui định bởi biến điều khiển.

Ví dụ : tính 10!

```
giaithua =1
for i = 1 to 10
    giaithua = giaithua * i
Next i
```


Lệnh lặp For...Next (tt)

Cú pháp 2 :

For Each *element* **In** *group*
[Statement]+
Next ***[****element]*

- *group* là 1 collection các đối tượng hay 1 dãy các phần tử. *element* là biến để chứa từng đối tượng hay từng phần tử trong group.
- Statement là 1 lệnh thực thi VB bất kỳ.
- thi hành các lệnh *[Statement]+* với từng phần tử trong 1 dãy hay với từng đối tượng trong 1 collection.

Ví dụ : hiển thị các đối tượng đồ họa của 1 ứng dụng

```
Dim objGraphObj As Object
...
For Each GraphObj In GraphObjList
    GraphObj.Draw          ' hiển thị đối tượng
Next GraphObj
```

8.5 Các lệnh lồng nhau

- Như ta đã thấy trong cú pháp của hầu hết các lệnh VB đều có chứa thành phần *Statement,* đây là 1 lệnh thực thi VB bất kỳ ⇒ ta gọi cú pháp định nghĩa lệnh VB là đệ qui ⇒ tạo ra các lệnh VB lồng nhau. Ta gọi cấp ngoài cùng là cấp 1, các lệnh hiện diện trong cú pháp của lệnh cấp 1 được gọi là lệnh cấp 2, các lệnh hiện diện trong cú pháp của lệnh cấp 2 được gọi là lệnh cấp 3,... Để dễ đọc, các lệnh cấp thứ i nên dóng hàng nhờ n ký tự Tab.

Ví dụ : đoạn chương trình tính ma trận tổng của 2 ma trận

```
Dim A(N,N) As Double, B(N,N) As Double
Dim C(N,N) As Double
For i = 1 to n          ' duyệt theo hàng
    For j = 1 to n      ' duyệt theo cột
        C(i,j) = A(i,j) + B(i,j)
    Next j
Next i
```


8.6 Vấn đề thoát đột ngột khỏi cấp điều khiển

- Như ta đã thấy trong cú pháp của hầu hết các lệnh VB đều có chứa thành phần *[Statement]+.* Theo trình tự thi hành thông thường, các lệnh bên trong này sẽ được thực thi tuần tự, hết lệnh này đến lệnh khác cho đến lệnh cuối, lúc này thì việc thi hành lệnh cha mới có thể kết thúc. Tuy nhiên trong 1 vài trạng thái thi hành đặc biệt, ta muốn thoát ra khỏi lệnh cha đột ngột chứ không muốn thực thi hết các lệnh con trong danh sách. Để phục vụ yêu cầu này, VB cung cấp lệnh Exit với cú pháp sau đây :

  Exit [**For** | **Do** | **Property** | **Sub** | **Function**]

- Lưu ý VB cho phép dùng Exit để thoát khỏi trực tiếp ra nhiều cấp. VB không cung cấp lệnh Exit If và Exit While để thoát khỏi lệnh If và lệnh While ⇒ dùng lệnh Do ... Loop thay thế lệnh While và/hoặc lệnh Goto (được trình bày sau).

Vấn đề thoát đột ngột khỏi cấp điều khiển (tt)

- Để thấy việc dùng các lệnh lồng nhau và yêu cầu cần thoát khỏi đột ngột 1 cấp điều khiển nào đó, ta hãy xem thủ tục sau, nó cho phép in ra tất cả các font chữ mà có thể dùng để hiển thị lên màn hình lẫn in ra máy in.

```
Private Sub Form_Click()
Dim objSFont As Object, objPFont As Object
' duyệt từng font màn hình
For Each objSFont In Screen.Fonts()
    ' duyệt từng font máy in
    For Each objPFont In Printer.Fonts()
        If objSFont = objPFont Then
            Print objSfont
        End If
    Next objPFont
Next objSFont
End Sub
```


Vấn đề thoát đột ngột khỏi cấp điều khiển (tt)

- Quan sát lệnh If ta thấy rằng điều kiện chỉ đúng tối đa 1 lần trong vòng lặp objPFont, do đó khi đã thỏa điều kiện rồi thì ta nên thoát khỏi vòng lặp này ngay (để thời gian chạy ít hơn ⇒ hiệu quả hơn).

```
Private Sub Form_Click()
Dim objSFont As Object, objPFont As Object
' duyệt từng font màn hình
For Each objSFont In Screen.Fonts()
    ' duyệt từng font máy in
    For Each objPFont In Printer.Fonts()
        If objSFont = objPFont Then
            Print objSfont
            Exit For        'thoát đột ngột khỏi vòng For trong cùng
        End If
    Next objPFont
Next objSFont
End Sub
```

Vấn đề thoát khỏi cấp điều khiển đột ngột (tt)

- Cũng thí dụ ở slide trước, nhưng nếu ta chỉ muốn in tên font chữ đầu tiên được dùng bởi cả màn hình và máy in, thì ta phải dùng lệnh Exit Sub sau khi đã in tên font đầu tiên này.

```
Private Sub Form_Click()
Dim SFont As objObject, objPFont As Object
' duyệt từng font màn hình
For Each objSFont In Screen.Fonts()
    ' duyệt từng font máy in
    For Each objPFont In Printer.Fonts()
        If objSFont = objPFont Then
            Print objSfont
            Exit Sub        'thoát đột ngột khỏi thủ tục
        End If
    Next objPFont
Next objSFont
End Sub
```


8.7 Lệnh gọi thủ tục

- Thủ tục là phương tiện phân chia code của module (class, form, standard) ra nhiều đơn vị nhỏ hơn để dễ quản lý và sử dụng. Đây là vấn đề khá lớn và sẽ được trình bày chi tiết trong chương kế.
- Ở đây chúng ta giới thiệu 1 vài ý tưởng ban đầu về thủ tục đủ để giới thiệu lệnh gọi (sử dụng) chúng. Thủ tục là 1 danh sách các lệnh VB thực hiện 1 chức năng rõ ràng (và thường đơn giản), các lệnh này được họp thành 1 đơn vị và được gán cho 1 tên nhận dạng, tên này nên gợi ý được chức năng của thủ tục (thí dụ ta đặt danh sách các lệnh VB tính cos của góc x trong 1 đơn vị và đặt tên cho nó là Cos).
- Để thủ tục có độ sử dụng cao, khi định nghĩa nó người ta kết hợp 1 danh sách tham số hình thức với nó. Mỗi tham số hình thức miêu tả 1 dữ liệu mà thủ tục sẽ xử lý khi thủ tục được thi hành. Các tham số là phương tiện trao đổi dữ liệu giữa lệnh gọi và code của thủ tục. Thí dụ ta kết hợp với thủ tục Cos 1 tham số là góc x, ta nói Cos (x) là thủ tục tính Cos của góc x.

Lệnh gọi thủ tục (tt)

- Sau khi đã định nghĩa thủ tục, ta có thể dùng (gọi) nó. Thủ tục chỉ được thi hành khi người ta gọi nó bằng lệnh gọi thủ tục. Cú pháp của lệnh gọi như sau :

  [Call] *name [arglist]*

Ví dụ : giả sử ta đã định nghĩa (viết) 1 thủ tục sau đây :

```
Private Sub Update_Display(d As Byte)
```

nó cho phép hiệu chỉnh giá trị Display sau khi người dùng ấn thêm ký số d. Như vậy khi người dùng ấn thêm ký số 5, ta sẽ thực hiện gọi thủ tục như sau :

```
Call Update_Display (5)
```

hay : Update_Display (5)

Lưu ý : Trong trường hợp gọi thủ tục không có bất kỳ tham số nào ta nên dùng thêm từ khóa "Call' để chương trình trong sáng, dễ đọc.

MÔN TIN HỌC

Chương 9

ĐỊNH NGHĨA THỦ TỤC & SỬ DỤNG

Nhắc lại cấu trúc tổ chức 1 chương trình

- Một chương trình thường cung cấp nhiều chức năng cho người dùng ⇒ Chương trình thường là 1 hệ thống phức tạp. Để dễ quản lý và xây dựng chương trình, người ta thường chia nó ra nhiều đơn vị nhỏ hơn. Hiện có 2 phương pháp chia nhỏ chương trình :
  - phương pháp có cấu trúc : chương trình được chia nhỏ thành nhiều module chức năng, mỗi module chứa nhiều điểm nhập (entry), mỗi điểm nhập cung cấp 1 dịch vụ (chức năng) rõ ràng, đơn giản nào đó. Ta gọi mỗi điểm nhập là thủ tục thực hiện chức năng tương ứng.
  - phương pháp hướng đối tượng : chương trình được chia nhỏ thành nhiều đối tượng, mỗi đối tượng chứa nhiều điểm nhập (entry), mỗi điểm nhập cung cấp 1 dịch vụ (chức năng) rõ ràng, đơn giản nào đó. Ta gọi mỗi điểm nhập là thủ tục thực hiện chức năng tương ứng.
- Tóm lại, dù dùng phương pháp chia nhỏ chương trình nào thì đơn vị chức năng nhỏ nhất mà người lập trình có thể xây dựng và dùng (gọi) lại nhiều lần trong chương trình là **thủ tục**.

9.1 Phân loại thủ tục trong VB

- Nếu ta phân tích chương trình theo cấu trúc thì chương trình VB là tập các standard module, trong mỗi module ta có thể định nghĩa n thủ tục khác nhau thuộc 1 trong 2 dạng :
  - thủ tục - Sub : 1 đoạn lệnh thực thi VB để thực hiện 1 chức năng rõ ràng, đơn giản nhưng không trả về giá trị kèm theo tên thủ tục.
  - hàm - Function : 1 đoạn lệnh thực thi VB để thực hiện 1 chức năng rõ ràng, đơn giản và trả về giá trị kèm theo tên hàm.
- Nếu ta phân tích chương trình theo hướng đối tượng thì chương trình VB là tập các form hay class module, trong mỗi module ta có thể định nghĩa n thủ tục khác nhau thuộc 1 trong 3 dạng :
  - thủ tục - Sub : 1 đoạn lệnh thực thi VB để thực hiện 1 chức năng rõ ràng, đơn giản nhưng không trả về giá trị kèm theo tên thủ tục.
  - hàm - Function : 1 đoạn lệnh thực thi VB để thực hiện 1 chức năng rõ ràng, đơn giản và trả về giá trị kèm theo tên hàm.
  - truy xuất thuộc tính - Property : 1 đoạn lệnh thực thi VB để đọc/ghi 1 thuộc tính tương ứng của đối tượng. Có 3 thủ tục loại này là Get, Set và Let.

Tầm vực sử dụng thủ tục trong VB

- Trong mỗi standard module, ta có thể xác định tầm vực sử dụng của từng thủ tục :
  - cục bộ trong module : dùng từ khóa Private trong lệnh định nghĩa thủ tục.
  - toàn cục trong chương trình : dùng từ khóa Public trong lệnh định nghĩa thủ tục.
- Trong mỗi form hay class module, ta có thể xác định tầm vực sử dụng của từng thủ tục :
  - cục bộ trong module (đối tượng) : dùng từ khóa Private trong lệnh định nghĩa thủ tục.
  - cục bộ trong Project : dùng từ khóa Friend trong lệnh định nghĩa thủ tục.
  - công cộng (ai dùng cũng được) : dùng từ khóa Public trong lệnh định nghĩa thủ tục. Các thủ tục công cộng của đối tượng được gọi là method để phân biệt với Sub/Function.
  - Về nguyên tắc, các thủ tục Property Get, Set và Let đều phải có tầm vực công cộng (dùng từ khóa Public).

9.2 Cú pháp định nghĩa hàm - Function

- Cú pháp để định nghĩa 1 hàm :
  [**Public** | **Private** | **Friend**] [**Static**] **Function** *name* [(*arglist*)] [**As** *type*]
  [*statements*]
  [*name* = *expression*]
  [**Exit Function**]
  [*statements*]
  [*name* = *expression*]
  End Function
- Dùng từ khóa **Public** để định nghĩa hàm có tầm vực toàn cục, nghĩa là bất kỳ lệnh nào của chương trình đều có thể gọi hàm Public.
- Dùng từ khóa **Friend** để định nghĩa method thuộc 1 class module nhưng chỉ có tầm vực cục bộ trong Project, nghĩa là chỉ có các lệnh trong cùng Project mới có thể gởi thông điệp đến hàm Friend của đối tượng đó, còn các lệnh ở ngoài Project thì không thấy hàm Friend của đối tượng này.

Cú pháp định nghĩa hàm - Function (tt)

- Dùng từ khóa **Private** để định nghĩa hàm có tầm vực cục bộ trong module, nghĩa là chỉ có các lệnh trong cùng module mới có thể gọi hàm Private trong module tương ứng.
- Dùng từ khóa **Static** để định nghĩa các biến cục bộ trong hàm đều là Static, nghĩa là giá trị của chúng vẫn tồn tại qua các lần gọi khác nhau đến hàm này.
- *[statements]* là danh sách các lệnh định nghĩa biến, hằng, kiểu cục bộ trong function và các lệnh thực thi miêu tả chính xác chức năng của hàm.
- Lệnh gán *name* **=** *expression* cho phép gán giá trị trả về cho lệnh gọi hàm.
- Lệnh **Exit Function** cho phép trả ngay điều khiển về lệnh gọi hàm này (thay vì thực thi tiếp các lệnh còn lại của hàm).

Cú pháp định nghĩa hàm - Function (tt)

- *arglist* là danh sách các tham số hình thức, mỗi tham số được cách nhau bởi dấu ',' và được định nghĩa theo cú pháp như sau :

  [**Optional**] [**ByVal** | **ByRef**] [**ParamArray**] *varname*[**( )**] [**As** *type*] [**=***defaultvalue*]
- Dùng từ khóa **Optional** để khai báo rằng tham số tương ứng là nhiệm ý trong lúc gọi hàm : truyền hay không cũng được. Trong trường hợp này ta nên dùng thêm thành phần [**=** *defaultvalue*] để xác định giá trị cần truyền nhiệm ý.
- Dùng từ khóa **ByRef** để khai báo việc truyền tham số bằng tham khảo, đây là chế độ truyền tham số nhiệm ý. Ngược lại dùng từ khóa **ByVal** để khai báo cơ chế truyền tham số bằng giá trị.
- Chỉ có thể dùng từ khóa **ParamArray** cho tham số cuối trong danh sách tham số, tham số này cho phép ta truyền bao nhiêu tham số cụ thể cũng được.

Thí dụ định nghĩa hàm

- Đoạn code sau định nghĩa hàm tính n! giai thừa theo giải thuật đệ qui :

```
Public Function giaithua(ByVal n As Long) As Long
    If n <= 0 Then   ' nếu n <=0 thì trả về -1
        giaithua = -1
        Exit Function
    End If
    If n = 1 Then     ' nếu n = 1 thì trả về kết quả là 1
        giaithua = 1
        Exit Function
    End If
    ' Nếu n > 1 thì tính theo công thức n! = n * (n-1)!
    giaithua = n * giaithua(n - 1)
End Function
```


9.3 Cú pháp định nghĩa thủ tục - Sub

- Cú pháp để định nghĩa 1 thủ tục Sub :
  [**Private** | **Public** | **Friend**] [**Static**] **Sub** *name* [(*arglist*)]
  [*statements*]
  [**Exit Sub**]
  [*statements*]
  End Sub
- Ý nghĩa của các từ khóa **Public, Private, Friend, Static** cũng như các thành phần *arglist*, **Exit Sub**, *statements* giống y như trong việc định nghĩa hàm mà chúng ta đã giới thiệu ở những slide trước.
- Sự khác biệt giữa hàm và thủ tục là hàm luôn trả về giá trị kết hợp với tên hàm, còn thủ tục thì không trả về trị kết hợp với tên thủ tục (nhưng nó vẫn có thể trả kết quả về thông qua các tham số truyền bằng tham khảo).
- Nếu quan sát kỹ, ta thấy các hàm xử lý sự kiện cho các đối tượng giao diện đều là Sub, chứ không phải là Function, do đó từ đây ta dùng đoạn câu "thủ tục xử lý sự kiện" thay cho "hàm xử lý sự kiện".

Cú pháp định nghĩa method Get thuộc tính đối tượng

- Cú pháp để định nghĩa 1 method Get :
  [**Public** | **Private** | **Friend**] [**Static**] **Property Get** *name* [**(***arglist***)**] [**As** *type*]
  [*statements*]
  [*name* **=** *expression*]
  [**Exit Property**]
  [*statements*]
  [*name* **=** *expression*]
  End Property
- Ý nghĩa của các từ khóa **Public, Private, Friend, Static** cũng như các thành phần *arglist*, **Exit Property**, *statements,* [*name* **=** *expression*] giống y như trong lệnh định nghĩa hàm mà chúng ta đã giới thiệu ở những slide trước.
- Method **Get** cho phép bên ngoài có thể đọc giá trị của 1 thuộc tính bên trong đối tượng nhưng dưới sự kiểm soát của đối tượng đó.

Cú pháp định nghĩa method Let thuộc tính đối tượng

- Cú pháp để định nghĩa 1 method Let :
  [**Public** | **Private** | **Friend**] [**Static**] **Property Let** *name* **(**[*arglist*,] *value***)**
  [*statements*]
  [**Exit Property**]
  [*statements*]
  End Property
- Ý nghĩa của các từ khóa **Public, Private, Friend, Static** cũng như các thành phần *arglist*, **Exit Property**, *statements* giống y như trong lệnh định nghĩa hàm mà chúng ta đã giới thiệu ở những slide trước.
- Method **Let** cho phép bên ngoài có thể gán giá trị mới cho 1 thuộc tính bên trong đối tượng nhưng dưới sự kiểm soát của đối tượng đó.

Cú pháp định nghĩa method Set thuộc tính đối tượng

- Cú pháp để định nghĩa 1 method Set :
  [**Public** | **Private** | **Friend**] [**Static**] **Property Set** *name* **(**[*arglist*,] *reference***)**
  [*statements*]
  [**Exit Property**]
  [*statements*]
  End Property
- ý nghĩa của các từ khóa **Public, Private, Friend, Static** cũng như các thành phần *arglist*, **Exit Property**, *statements* giống y như trong lệnh định nghĩa hàm mà chúng ta đã giới thiệu ở những slide trước.
- Method **Set** cho phép bên ngoài có thể gán tham khảo cho 1 thuộc tính bên trong đối tượng nhưng dưới sự kiểm soát của đối tượng đó.
- Sự khác biệt giữa method Let và Set là Let gán giá trị thuộc 1 kiểu cổ điển, còn Set gán tham khảo vào 1 thuộc tính có kiểu là class đối tượng.

9.4 Gọi thủ tục

- Sau khi thủ tục đã được định nghĩa, ta có thể sử dụng (gọi) nó nhờ lệnh gọi thủ tục. Cú pháp gọi thủ tục đã được miêu tả trong slide 216 (chương 8). Do Function là dạng thủ tục có trả về kết quả kết hợp với tên hàm nên lệnh gọi hàm thường được dùng trong 1 biểu thức (lệnh gọi hàm là biểu thức cơ bản để cấu thành biểu thức phức tạp hơn).
- Thí dụ, giả sử ta đã định nghĩa hàm tính n! tên là giaithua(n) thì ta có thể gọi nó như sau :

```
n = 8
MsgBox (n & "! = " & giaithua(n))
```

- Thí dụ, giả sử ta đã định nghĩa thủ tục hoán vị 2 số nguyên tên là Hoanvi(a,b) thì ta có thể gọi nó như sau :

```
n = 8
m = 4
Call Hoanvi (n,m)          ' hoặc Hoanvi n,m
' Lúc này n = 8 và m = 4
```

9.5 Cơ chế truyền tham số

- Các tham số trong lệnh định nghĩa thủ tục được gọi là tham số hình thức. Các tham số (thường là biểu thức) trong lệnh gọi thủ tục được gọi là tham số thực. Nguyên tắc gọi thủ tục là :
  - số lượng các tham số thực phải bằng số lượng các tham số hình thức.
  - và kiểu của từng tham số thực trong lệnh gọi thủ tục phải trùng (hay tương thích) với kiểu của tham số hình thức tương ứng trong lệnh định nghĩa thủ tục.
- Lệnh gọi thủ tục sẽ truyền tham số thực trong lệnh gọi cho thủ tục rồi khởi động thủ tục chạy để xử lý tham số thực vừa nhận được. Theo thời gian, thủ tục sẽ được gọi nhiều lần, mỗi lần với danh sách tham số thực cụ thể.
- Có 2 cơ chế truyền tham số cho thủ tục tại thời điểm gọi thủ tục : truyền giá trị (nội dung của tham số) hay truyền tham khảo (địa chỉ - vị trí bộ nhớ của tham số). Mỗi cơ chế truyền tham số có tính chất riêng mà ta sẽ trình bày kỹ trong các slide kế tiếp :

Cơ chế truyền tham số (tt)

- Dùng từ khóa **ByVal** kết hợp với tham số hình thức để khai báo nó được truyền bằng giá trị. Khi gọi thủ tục, **giá trị của tham số thực** sẽ được truyền cho thủ tục cần thực thi. Nhờ cách truyền tham số này mà thủ tục cần thực thi sẽ không thể truy xuất dữ liệu của thủ tục gọi. Tuy nhiên cách truyền bằng giá trị chỉ thích hợp cho các tham số IN (truyền từ phần tử gọi đến thủ tục cần gọi) có kiểu vô hướng (scalar).
- Để truyền hiệu quả tham số có nội dung chiếm nhiều ô nhớ hay để nhận kết quả ta sẽ phải dùng cơ chế truyền bằng tham khảo (địa chỉ). Để định nghĩa tham số hình thức được truyền bằng tham khảo, ta dùng từ khóa **ByRef** kết hợp với tham số hình thức đó. Khi gọi thủ tục, **địa chỉ của tham số thực** sẽ được truyền cho thủ tục cần thực thi. Với đặc điểm này, tham số thực phải là biến chứ không thể là biểu thức.
- Lưu ý rằng nếu ta không dùng từ khóa ByRef hay ByVal kết hợp với tham số hình thức thì default nó được truyền bằng tham khảo.

Cơ chế truyền tham số (tt)

```
// version truyền bằng giá trị
Private Sub Hoanvi1(ByVal x As Integer, ByVal y As Integer)
Dim tmp As Integer
   tmp = x
   x = y
   y = tmp
End Sub

// version truyền bằng tham khảo
Private Sub Hoanvi2(ByRef x As Integer, ByRef y As Integer)
Dim tmp As Integer
   tmp = x
   x = y
   y = tmp
End Sub

// version truyền bằng tham khảo
Private Sub Hoanvi3(x As Integer, y As Integer)
Dim tmp As Integer
   tmp = x
   x = y
   y = tmp
End Sub
```


Cơ chế truyền tham số (tt)

Hãy khảo sát kỹ 3 thủ tục hoán vị dữ liệu trong slide trước. Bây giờ hãy chú ý tới việc sử dụng chúng và kết quả đạt được :

```
...
Dim intN As Integer
Dim intM As Integer
   intN = 4
   intM = 8
   Call Hoanvi1(intN, intM)  ' kết quả intN = 4 và intM = 8 (không đổi)
   Call Hoanvi2(intN, intM)  ' kết quả intN = 8 và intM = 4 (đã hoán vị được)
   Call Hoanvi3(intN, intM)  ' kết quả intN = 4 và intM = 8 (đã hoán vị được)
...
```

9.6 Các thủ tục định sẵn của VB

- Về nguyên tắc, người lập trình phải định nghĩa thủ tục (Sub, Function, Property) trước khi có thể sử dụng lại (gọi) nó. Tuy nhiên, VB đã định nghĩa rất nhiều thủ tục dạng Sub, Function để thực hiện các chức năng rất phổ biến, người lập trình có thể gọi chúng bất cứ khi nào cần thiết. Ta gọi các thủ tục này là các thủ tục định sẵn của VB.
- Nếu chưa đòi hỏi độ chính xác cao, người ta còn gọi các thủ tục định sẵn của VB là các lệnh thực thi.
- Sau đây ta chúng ta hãy làm quen với 1 số thủ tục thường dùng.

Hàm hiển thị form thông báo

- Cú pháp **MsgBox(***prompt [, buttons] [, title] [, helpfile, context]***)** trong đó :
  - *prompt* là biểu thức chuỗi miêu tả thông báo cần hiển thị.
  - *buttons* là biểu thức số miêu tả số lượng và loại button được hiển thị trong thông báo, nhiệm ý là 0 nghĩa là chỉ có button Ok được hiển thị.
  - *title* là biểu thức chuỗi miêu tả title bar của form thông báo.
  - *helpfile* là biểu thức chuỗi miêu tả đường dẫn file Help được dùng với form thông báo (theo cơ chế context-sensitive Help).
  - *context* là biểu thức số miêu tả chỉ số của "topic" cần dùng trong file Help
- Thường để gọi dễ dàng hàm MsgBox, ta chỉ cần miêu tả tham số *prompt* bắt buộc.

Hàm hiển thị form nhập liệu (dạng chuỗi)

- Cú pháp **InputBox (***prompt* [,*title*] [,*default*] [,*xpos*] [,*ypos*] [,*helpfile*,*context*]**)**

  trong đó :
  - *prompt, title, helpfile, context* là các tham số với ý nghĩa y như trong hàm MsgBox.
  - *xpos, ypos* là biểu thức số miêu tả tọa độ (x,y) của điểm trên trái của form thông báo trong màn hình. Nếu không được khai báo, form thông báo sẽ được chỉnh vị trí tự động (giữa màn hình).
  - *default* là biểu thức chuỗi miêu tả giá trị default của chuỗi được nhập.
- Thường để gọi dễ dàng hàm InputBox, ta chỉ cần miêu tả tham số *prompt* bắt buộc.

Hàm chuyển đổi kiểu

- VB cung cấp các hàm sau để ta có thể chuyển giá trị từ kiểu nào đó về kiểu xác định :

  CBool (*expression***)** : chuyển trị của biểu thức về kiểu Boolean
  CByte (*expression***)** : chuyển trị của biểu thức về kiểu Byte
  CCur (*expression***)** : chuyển trị của biểu thức về kiểu Currency
  CDate (*expression***)** : chuyển trị của biểu thức về kiểu Date
  CDbl (*expression***)** : chuyển trị của biểu thức về kiểu Double
  CDec (*expression***)** : chuyển trị của biểu thức về kiểu Decimal
  CInt (*expression***)** : chuyển trị của biểu thức về kiểu Integer
  CLng (*expression***)** : chuyển trị của biểu thức về kiểu Long
  CSng (*expression***)** : chuyển trị của biểu thức về kiểu Single
  CStr (*expression***)** : chuyển trị của biểu thức về kiểu String
  CVar (*expression***)** : chuyển trị của biểu thức về kiểu Variant

Các hàm thư viện liên kết động

- Trong code VB, ngoài việc gọi các thủ tục được định nghĩa trong Project và các thủ tục định sẵn, người lập trình còn có thể gọi các hàm trong các thư viện liên kết động.
- 1 thư viện liên kết động có dạng *.dll thường được xây dựng bằng ngôn ngữ VC++ và chứa 1 danh sách các hàm dịch vụ. Khi chương trình VB gọi 1 hàm trong file thư viện DLL, file được nạp vào bộ nhớ và hàm được liên kết vào vùng nhớ của chương trình để chương trình có thể gọi được hàm cần gọi. Các hàm thư viện DLL được sử dụng chung cho mọi phần mềm đang chạy, nghĩa là chỉ có 1 bản (copy) của hàm thư viện DLL trong bộ nhớ máy tính để phục vụ cho mọi ứng dụng gọi nó.
- Ta có thể coi Windows như 1 thư viện phần mềm DLL lớn, thư viện này cung cấp rất nhiều hàm dịch vụ khác nhau, người ta gọi các hàm này là các hàm API (Application Programming Interface). Chương trình VB có thể gọi bất kỳ hàm nào trong thư viện này theo cơ chế liên kết động như đã trình bày ở trên.
- Trước khi 1 hàm DLL được dùng trong module VB nào đó, ta cần khai báo đặc tả hàm DLL này nhờ lệnh **Declare** của VB với cú pháp được trình bày trong slide 153 (chương 6).

MÔN TIN HỌC

Chương 10

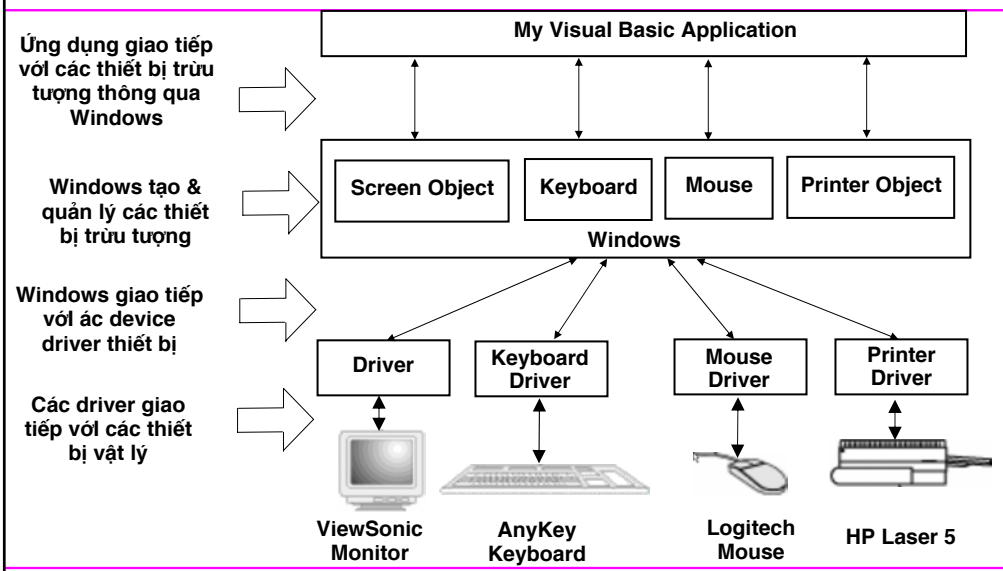
TƯƠNG TÁC GIỮA NGƯỜI DÙNG & CHƯƠNG TRÌNH

10.1 Tổng quát về tương tác giữa người dùng & chương trình
10.2 Giao tiếp với b2n phím.
10.3 Giao tiếp với chuột
10.4 Vẽ văn bản và đồ họa lên đối tượng giao diện
10.5 Vấn đề in ấn trong VB

10.1 Tổng quát về tương tác giữa người dùng & chương trình

- Trong lúc chương trình chạy, nó thường tương tác với người dùng. Sự tương tác gồm 2 hoạt động chính :
  - chờ nhận dữ liệu do người dùng cung cấp hay chờ nhận lệnh của người dùng để thực thi 1 chức năng nào đó.
  - hiển thị thông báo và/hoặc kết quả tính toán ra màn hình/máy in để người dùng biết và sử dụng.
- Sự tương tác giữa người dùng và máy tính được thực hiện thông qua các thiết bị nhập/xuất (thiết bị I/O - input/output) như bàn phím/chuột để nhập dữ liệu hay lệnh, màn hình/máy in để xuất kết quả hay thông báo...
- Hiện có hàng trăm hãng khác nhau chế tạo thiết bị I/O cho máy PC, mỗi hãng chế tạo rất nhiều model của cùng 1 thiết bị (thí dụ hãng HP đã chế rất nhiều model máy in phun mực, máy in laser,...). Mỗi model thiết bị của từng hãng có những tính chất vật lý riêng và khác với các model khác.
- Để giúp người lập trình truy xuất các thiết bị I/O dễ dàng, độc lập với tính chất phần cứng của thiết bị, HĐH Windows và VB đã che dấu mọi tính chất phần cứng của các thiết bị và cung cấp cho người lập trình 1 giao tiếp sử dụng duy nhất, độc lập với thiết bị.

Kiến trúc tương tác giữa người dùng & ứng dụng VB



Kiến trúc tương tác giữa người dùng & ứng dụng VB (tt)

- Xem hình vẽ của slide trước (miêu tả kiến trúc tương tác giữa người dùng & ứng dụng VB), ta thấy :
  - cấp thấp nhất là các **thiết bị phần cứng**, mỗi thiết bị có tính chất riêng và khác với các thiết bị khác (ngay cả cùng loại, cùng hãng nhưng khác model).
  - cấp **device driver** điều khiển và giao tiếp trực tiếp với phần cứng nhưng che dấu mọi tính chất chi tiết của phần cứng, nó cung cấp cho cấp trên 1 giao tiếp sử dụng phần cứng độc lập với tính chất phần cứng đó ⇒ Mỗi model thiết bị phần cứng của 1 hãng cần có device driver riêng.
  - **cấp HĐH** xử lý các chức năng luận lý (đệm dữ liệu, xử lý sai,...) trước khi nhờ device driver giao tiếp trực tiếp với phần cứng. Windows che dấu các loại phần cứng và tạo ra những thiết bị trừu tượng để ứng dụng truy xuất chúng dễ dàng và độc lập với loại thiết bị (đối tượng Printer, Screen, Mouse, Keyboard).
  - VB tạo ra những **đối tượng giao diện** cao cấp và dễ dùng : mỗi đối tượng giao diện (form, window, listbox,...) đều có thể giao tiếp trực tiếp với người dùng để nhập/xuất dữ liệu, chờ nhận sự kiện hay chủ động thông báo cho user.

Kiến trúc tương tác giữa người dùng & ứng dụng VB (tt)

- Sau khi đã biết kiến trúc giao tiếp I/O của ứng dụng VB, khi cần giao tiếp với người dùng, ta nên :
  - dùng các đối tượng giao diện cao cấp (định sẵn của VB hay ActiveX Control).
  - trong 1 số trường hợp cần thiết ta sẽ dùng các đối tượng của Windows như Printer và Screen.
  - trong 1 số trường hợp tối cần thiết ta mới gọi các hàm trong giao tiếp của device driver.
  - và tuyệt đối không nên truy xuất trực tiếp phần cứng thiết bị I/O vì rất khó khăn, không an toàn, dễ bị tranh chấp với các ứng dụng chạy đồng thời.
- Tương tác với người dùng thông qua các đối tượng giao diện được thực hiện như sau :
  - nhập liệu/nhận lệnh thông qua các thủ tục xử lý sự kiện của phần tử giao diện tương ứng.
  - xuất kết quả/thông báo bằng cách gán kết quả vào thuộc tính tương ứng của đối tượng giao diện hay dùng các method vẽ đồ họa tổng quát.

10.2 Giao tiếp với keyboard qua các đối tượng giao diện

- Mặc dù có thể có nhiều phần tử giao diện cùng được hiển thị trên màn hình tại từng thời điểm nhưng chỉ có 1 phần tử giao diện được giao tiếp với thiết bị I/O, ta gọi phần tử giao diện này là '**active**' hay được '**focus**'.
- Liên quan đến việc ấn thả 1 phím, VB sẽ tạo ra 3 sự kiện sau đây và gởi về cho phần tử được 'focus' hiện hành :
  - **KeyDown** : sự kiện xảy ra khi người sử dụng bấm (ấn xuống) bất kỳ một phím nào trên bàn phím.
  - **KeyUp** : sự kiện xảy ra khi người sử dụng thả phím vừa ấn ra.
  - **KeyPress**: sự kiện xảy ra khi người sử dụng ấn/thả bất kỳ một phím nào trên bàn phím mà tạo ra được 1 ký tự ANSI.

Thủ tục xử lý sự kiện KeyDown, KeyUp & KeyPress

- Thủ tục có dạng sau :
  Private Sub *ControlName*_**KeyDown (KeyCode as Integer, Shift as Integer).**

và **Private Sub** *ControlName*_**KeyUp (KeyCode as Integer, Shift as Integer).**
  trong đó :
  - *ControlName* là tên của điều khiển nhận sự kiện keydown/keyup.
  - KeyCode là mã "virtual code" của phím được ấn/thả.
  - Shift là giá trị miêu tả trạng thái ấn giữ các phím điều khiển (là một dãy bit với bit 0 cho phím SHIFT, bit 1 cho phím CTRL, bit 2 cho phím ALT).
- Thủ tục KeyPress có dạng sau :
  Private Sub *ControlName*_**KeyPress (KeyAscii As Integer)**
  trong đó :
  - *ControlName* là tên của điều khiển nhận sự kiện keypress.
  - KeyAscii là mã ký tự ANSI của phím được ấn/thả.
- Mỗi đối tượng có thủ tục xử lý biến cố riêng, thủ tục này cũng là method của đối tượng tương ứng.

Thí dụ thủ tục xử lý biến cố KeyDown của 1 textbox

```
Private Sub Text1_KeyDown (KeyCode As Integer, Shift As Integer)
Dim ShiftDown, AltDown, CtrlDown, Txt
ShiftDown = (Shift And vbShiftMask) > 0
AltDown = (Shift And vbAltMask) > 0
CtrlDown = (Shift And vbCtrlMask) > 0
    If KeyCode = vbKeyF2 Then    ' Display key combinations.
        If ShiftDown And CtrlDown And AltDown Then
            Txt = "SHIFT+CTRL+ALT+F2."
        ElseIf ShiftDown And AltDown Then
            Txt = "SHIFT+ALT+F2."
        ElseIf ShiftDown And CtrlDown Then
            Txt = "SHIFT+CTRL+F2."
        ElseIf CtrlDown And AltDown Then
            Txt = "CTRL+ALT+F2."
        ElseIf ShiftDown Then
            Txt = "SHIFT+F2."
        ElseIf CtrlDown Then
            Txt = "CTRL+F2."
        ElseIf AltDown Then
            Txt = "ALT+F2."
        ElseIf SHIFT = 0 Then
            Txt = "F2."
        End If
        Text1.Text = "You pressed " & Txt
    End If
End Sub
```


Dùng thuộc tính KeyPreview

- Thường 1 form giao diện (hộp thoại) chứa nhiều điều khiển bên trong nó ⇒ Khi thao tác phím trên 1 điều khiển trong form thì sự kiện sẽ gởi cho điều khiển hay form ? Để qui định cụ thể điều này, VB cung cấp thuộc tính KeyPreview cho form, ta có thể xem/hiệu chỉnh giá trị của nó nhờ lệnh gán :

  FormName.**KeyPreview [=** *boolean_expr*]

  trong đó :

  - *FormName* là tên của form liên quan.
  - *boolean_expr* là biểu thức luận lý có giá trị True/False.
- Khi ta gán trị luận lý vào thuộc tính của form thì nếu :
  - trị = True thì form sẽ nhận và xử lý biến cố trước rồi mới tới điều khiển.
  - trị = False thì điều khiển nhận và xử lý biến cố, còn form thì không.

10.3 Giao tiếp với chuột thông qua các đối tượng giao diện

- Tương tự như bàn phím, khi người dùng thao tác chuột, VB sẽ tạo ra 1 trong 5 biến cố sau đây và gởi về cho phần tử được 'focus' hiện hành :
  - **MouseMove** : sự kiện xảy ra khi người sử dụng di chuyển chuột.
  - **MouseDown** : sự kiện xảy ra khi người sử dụng ấn bất kỳ nút nào trên chuột (tùy loại chuột mà nó có 1/2/3 nút).
  - **MouseUp** : sự kiện xảy ra khi người sử dụng thả nút vừa ấn ra.
  - **Click** : sự kiện xảy ra khi người sử dụng ấn và thả chuột.
  - **DblClick** : sự kiện xảy ra khi người sử dụng 'Click' chuột liên tục hai lần trong 1 thời gian đủ nhỏ (do người dùng qui định chung cho môi trường Windows).

Thủ tục xử lý sự kiện MouseDown & MouseUp

- Thủ tục có dạng sau :

  Private Sub *ControlName*_**MouseDown (Button As Integer, Shift As Integer, x As Single, y As Single)**

và **Private Sub** *ControlName*_**MouseUp (Button As Integer, Shift As Integer, x As Single, y As Single)**

trong đó :

- *ControlName* là tên của điều khiển nhận sự kiện MouseDown/MouseUp.
- Button là giá trị miêu tả trạng thái các nút của chuột được ấn/thả (là một dãy các bit với bit 0 cho nút trái, bit 1 cho nút phải và bit 2 cho nút giữa).
- Shift là giá trị miêu tả trạng thái ấn giữ các phím điều khiển (là một dãy bit với bit 0 cho phím SHIFT, bit 1 cho phím CTRL, bit 2 cho phím ALT).
- x, y miêu tả tọa độ (x,y) của vị trí chuột được ấn/thả trên màn hình.

Thủ tục xử lý sự kiện MouseMove

- Thủ tục có dạng sau :

  Private Sub *ControlName*_**MouseMove (Button As Integer, Shift As Integer, x As Single, y As Single)**

  trong đó :
  - *ControlName* là tên của điều khiển nhận sự kiện MouseDown/MouseUp.
  - Button là giá trị miêu tả trạng thái các nút của chuột được ấn/thả (là một dãy các bit với bit 0 cho nút trái, bit 1 cho nút phải và bit 2 cho nút giữa).
  - Shift là giá trị miêu tả trạng thái ấn giữ các phím điều khiển (là một dãy bit với bit 0 cho phím SHIFT, bit 1 cho phím CTRL, bit 2 cho phím ALT).
  - x, y miêu tả tọa độ (x,y) của vị trí chuột hiện hành trên màn hình.

Thủ tục xử lý sự kiện Click & DblClick

- Thủ tục có dạng sau :

  Private Sub *ControlName*_**Click ()**

và **Private Sub** *ControlName*_**DblClick ()**

  trong đó :
  - *ControlName* là tên của điều khiển nhận sự kiện Click/DblClick.
  - Thủ tục xử lý sự kiện Click và DblClick không có tham số để xác định vị trí ấn chuột hay nút chuột nào đã được ấn. Trong trường hợp cần các thông tin phụ này để xử lý chi li hơn, bạn nên dùng thủ tục xử lý sự kiện MouseDown hay MouseUp.

Thí dụ thủ tục xử lý các sự kiện chuột

```
' biến qui định trạng thái vẽ/không vẽ
Dim PaintNow As Boolean

'Khởi động thông số vẽ
Private Sub Form_Load ()
   DrawWidth = 10      ' Use wider brush.
   ForeColor = RGB(0, 0, 255)   ' Set drawing color.
End Sub

Private Sub Form_MouseDown (Button As Integer, Shift As Integer, X As
Single, Y As Single)
   PaintNow = True   ' Enable painting.
End Sub

Private Sub Form_MouseUp (Button As Integer, Shift As Integer, X As Single, Y As Single)
   PaintNow = False   ' Disable painting.
End Sub

Private Sub Form_MouseMove (Button As Integer, Shift As Integer, X As Single, Y As
Single)
   If PaintNow Then
      PSet (X, Y)      ' Draw a point.
   End If
End Sub
```


10.4 Vẽ văn bản và đồ họa lên đối tượng giao diện

- Các điều khiển thường chứa thuộc tính Text, Caption hay Value để cho phép chương trình truy xuất (đọc/ghi) các thuộc tính này. Tuy nhiên những đối tượng giao diện phức hợp như Form, PictureBox, Printer có thể hiển thị nhiều nội dung chi tiết ở dạng văn bản, đồ họa hay ảnh bitmap bất kỳ. Để hiển thị các nội dung này, ta dùng các method sau của các đối tượng tương ứng :
  - **Cls** : xóa toàn bộ nội dung hiển thị trước đó của đối tượng.
  - **Print** : hiển thị 1 hay nhiều chuỗi văn bản.
  - **PSet** : hiển thị 1 điểm pixel với 1 màu xác định.
  - **Point** : trả về giá trị màu của 1 điểm pixel.
  - **Line** : vẽ 1 đoạn thẳng hay 1 hộp hình chữ nhật.
  - **Circle** : cẽ 1 hình tròn, ellipse hay cung.
  - **PaintPicture** : vẽ 1 ảnh bitmap đã có vào đối tượng.
- Các slide còn lại diễn tả chi tiết các method trên cùng các thí dụ về việc dùng chúng.

Thủ tục Print : xuất chuỗi ra thiết bị xuất luận lý

- Thủ tục có dạng sau :
  [*objName*.]**Print [Spc(**n**)** | **Tab(**n**)** | *expression charpos*]*
  trong đó :
  - *objName* là tên của đối tượng nhận kết quả vẽ (Printer, Form, PictureBox), default là form hiện hành.
  - Spc(n) qui định n ký tự trống được in ra.
  - Tab(n) qui định n ký tự Tab được in ra, mỗi Tab đưa pointer in qua phải thêm 1 cột (vị trí các cột được qui định trước).
  - *expression* là biểu thức chuỗi hay số cần in.
  - *charpos* qui định vị trí in dữ liệu kế tiếp. Nếu *charpos* = ";" thì dữ liệu in kế tiếp sẽ được in liền ngay. Nếu *charpos* = "," thì sẽ thêm 1 Tab trước khi in dữ liệu kế. Nếu không có *charpos* sau cùng thì vị trí in sẽ dời xuống đầu dòng kế tiếp.
- Thông tin về font chữ phải được thiết lập trước thủ tục Print thông qua các thuộc tính sau của đối tượng vẽ : FontName, FontSize, FontItalic, FontBold,...
- Nên thiết lập thuộc tính CurrentX, CurrentY để qui định rõ ràng vị trí in của mỗi lệnh Print.

Thí dụ về việc dùng thủ tục Print

- Đoạn code sau đây sẽ hiển thị 3 hàng văn bản trong hộp thoại About Box được chỉnh giữa :

```
Const strAbout1 = "Trinh MiniIE"
Const strAbout2 = "Version 1.0"
Const strAbout3 = "Written by : Nguyen Van Hiep"
Private Sub Form_paint()
        ScaleMode = vbPixels
        ' Xác định vị trí để chuỗi strAbout1 nằm giữa hộp thoại
        CurrentX = (ScaleWidth - TextWidth(strAbout1)) / 2
        CurrentY = 40
        Print strAbout1
        ' Xác định vị trí để chuỗi strAbout2 nằm giữa hộp thoại
        CurrentX = (ScaleWidth - TextWidth(strAbout2)) / 2
        CurrentY = 60
        Print strAbout2
        ' Xác định vị trí để chuỗi strAbout3 nằm giữa hộp thoại
        CurrentX = (ScaleWidth - TextWidth(strAbout3)) / 2
        CurrentY = 80
        Print strAbout3
End Sub
```

Function Format : chỉnh dạng dữ liệu trước khi in

- Ta thường muốn format dữ liệu số hay ngày tháng theo yêu cầu riêng trước khi in nó ra. VB hỗ trợ chức năng này thông qua hàm Format có cú pháp sau :
  Format (*expression* [,*format*[,*firstdayofweek*[,*firstweekofyear*]]])
  trong đó :
  - *expression* là biểu thức số hay ngày tháng cần format.
  - *format* là chuỗi ký tự định dạng hay tên gợi nhớ miêu tả chuỗi định dạng sẵn có của VB.
  - *firstdayofweek* và *firstweekofyear* qui định ngày đầu trong tuần và tuần đầu trong năm cần cho định dạng dữ liệu ngày tháng.
- Một số ký tự thường dùng trong chuỗi định dạng :

0	miêu tả vị trí ký số, nếu số không hiển thị hết vùng định đạng thì thêm số 0 trước và sau giá trị số cho đầy vùng định dạng.
#	miêu tả vị trí ký số, không in số 0 đi trước và sau giá trị số.
.	miêu tả vị trí dấu ngăn đơn vị (qui định bởi locale của Windows)
,	miêu tả vị trí dấu ngăn đơn vị ngàn (qui định bởi locale).
- + % ( ) space	miêu tả chỉnh xác ký tự tương ứng.

Thí dụ về hàm Format

- **Thí dụ về chỉnh dạng dữ liệu số** :

Format syntax	Result
Format(8315.4, "00000.00")	08315.40
Format(8315.4, "#####.##")	8315.4
Format(8315.4, "##,##0.00")	8,315.40
Format(315.4,"$##0.00")	$315.40

- **Thí dụ về chỉnh dạng dữ liệu ngày tháng** :

Format(Now, "m/d/yy")	1/27/93
Format(Now, "dddd, mmmm dd, yyyy")	Wednesday, January 27, 1993
Format(Now, "d-mmm")	27-Jan
Format(Now, "mmmm-yy")	January-93
Format(Now, "hh:mm AM/PM")	07:18 AM
Format(Now, "h:mm:ss a/p")	7:18:00 a
Format(Now, "d-mmmm h:mm")	3-January 7:18

Thủ tục PSet : vẽ điểm trên thiết bị xuất luận lý

- Thủ tục có dạng sau :

  [*objName*.]**PSet [Step] (***x, y***), [***color***]**

  trong đó :
  - *objName* là tên của đối tượng nhận kết quả vẽ (Printer, Form, PictureBox), default là form hiện hành.
  - (x,y) miêu tả tọa độ của vị trí điểm cần vẽ trên thiết bị luận lý. Nếu từ khóa **Step** được dùng thì (x,y) là tọa độ tương đối so với vị trí hiện hành được xác định bởi 2 thuộc tính CurrentX, CurrentY của đối tượng vẽ. Nếu từ khóa Step không được dùng thì (x,y) là tọa độ so với điểm gốc (0,0).
  - *color* là giá trị kiểu Long miêu tả màu vẽ theo hệ màu RGB (hoặc dùng hàm QBColor() hoặc dùng hàm RGB(r,g,b) để xác định màu vẽ).

Thủ tục Line : vẽ đoạn thẳng/box trên thiết bị xuất luận lý

- Thủ tục có dạng sau :
  [*objName*.]**Line [Step] (***x1, y1***) - [Step] (***x2, y2***), [***color***], [B][F]**
  trong đó :
  - *objName* là tên của đối tượng nhận kết quả vẽ (Printer, Form, PictureBox), default là form hiện hành.
  - *(x1,y1)* miêu tả tọa độ điểm đầu của đoạn thẳng cần vẽ trên đối tượng nhận kết quả. Nếu từ khóa **Step** được dùng trước tọa độ (*x1,y1*) thì nó là tọa độ tương đối so với vị trí hiện hành được xác định bởi 2 thuộc tính CurrentX, CurrentY của đối tượng nhận kết quả. Nếu từ khóa Step không được dùng thì *(x1,y1)* là tọa độ so với điểm gốc (0,0).
  - (*x2,y2*) miêu tả tọa độ điểm cuối của đoạn thẳng cần vẽ. Ý nghĩa của (*x2,y2*) cũng giống như (*x1,y1*).
  - *color* là giá trị kiểu Long miêu tả màu vẽ theo hệ màu RGB.
  - nếu không có thông số **B** thì thủ tục Line sẽ vẽ đoạn thẳng qua 2 điểm.
  - nếu có thông số **B**, thủ tục Line sẽ vẽ hình chữ nhật mà 2 đỉnh chéo được xác định bởi 2 điểm. Trong trường hợp này nếu có thông số **F**, hình chữ nhật sẽ được tô cùng màu với màu vẽ, ngược lại thuộc tính FillColor và FillStyle của đối tượng nhận kết quả sẽ qui định màu được tô.

Thủ tục Circle : vẽ hình tròn/ellipse trên thiết bị xuất luận lý

- Thủ tục có dạng sau :
  [*objName*.]**Circle [Step] (***x, y***),** *radius*, [*color, start, end, aspect*]
  trong đó :
  - *objName* là tên của đối tượng nhận kết quả vẽ (Printer, Form, PictureBox), default là form hiện hành.
  - (*x,y*) miêu tả tọa độ tâm điểm của vòng tròn/ellipse/arc cần vẽ trên đối tượng nhận kết quả. Nếu từ khóa **Step** được dùng trước tọa độ (*x,y*) thì (*x,y*) là tọa độ tương đối so với vị trí hiện hành được xác định bởi 2 thuộc tính CurrentX, CurrentY của đối tượng nhận kết quả. Nếu từ khóa Step không được dùng thì (*x,y*) là tọa độ so với điểm (0,0).
  - *radius* miêu tả bán kính.
  - *color* là giá trị kiểu Long miêu tả màu vẽ theo hệ màu RGB.
  - *start, end* miêu tả góc xác định điểm đầu và cuối của arc theo đơn vị radian (default điểm đầu là 0 và điểm cuối là 2π).
  - *aspect* miêu tả tỉ lệ kích thước dọc/ngang của ellipse (default là 1 để vẽ vòng tròn).

Các thuộc tính qui định thông số vẽ

- Khi ta gọi các method vẽ PSet, Line, Circle, PaintPicture trên 1 đối tượng vẽ nào đó (Printer, Form, PictureBox) thì các method này dùng các thuộc tính sau để qui định thông số vẽ của chúng :
  - CurrentX, CurrentY miêu tả tọa độ điểm hiện hành, nó được dùng làm gốc tọa độ cho các điểm vẽ nếu có dùng từ khóa Step kèm theo điểm vẽ đó.
  - FillStyle, FillColor xác định mẫu tô và màu tô các phần tử có diện tích (box, circle).
  - BackColor xác định màu nền của đối tượng.
  - ForeColor xác định màu để hiển thị text hay vẽ biên các phần tử (line, box, circle).
  - DrawMode xác định cách thức vẽ (vbBlackness, vbWhiteness, vbInvert...).
  - DrawStyle xác định mẫu vẽ của đường vẽ (line, box, circle).
  - DrawWidth xác định độ dày của đường vẽ (line, box, circle).
- Ta có thể đọc/hiệu chỉnh lại giá trị các thuộc tính theo yêu cầu.

Chi tiết về thuộc tính qui định đơn vị tính kích thước

- Thuộc tính ScaleMode miêu tả đơn vị tính kích thước với qui định sau :

Constant	Setting	Description
vbUser	0	Indicates that one or more of the ScaleHeight, ScaleWidth, ScaleLeft, and ScaleTop properties are set to custom values.
vbTwips	1	(Default)Twip (1440 twips per logical inch; 567 twips per logical centimeter).
vbPoints	2	Point (72 points per logical inch).
vbPixels	**3**	**Pixel (smallest unit of monitor or printer resolution).**
vbCharacters	4	Character (horizontal = 120 twips per unit; vertical = 240 twips per unit).
vbInches	5	Inch.
vbMillimeters	6	Millimeter.
vbCentimeters	7	Centimeter.
vbHimetric	8	HiMetric
vbContainerPosition	9	Units used by the control's container to determine the control's position.
vbContainerSize	10	Units used by the control's container to determine the control's size.

Chi tiết về thuộc tính miêu tả màu

- Mỗi màu ở chế độ TrueColor được tổng hợp từ 3 thành phần màu cơ bản Red - Green - Blue. Trọng số của mỗi thành phần màu được miêu tả bởi 1 giá trị Byte (từ 0 đến 255). Xác định 1 màu là xác định 3 thành phần màu của nó.
- Các thuộc tính BackColor, ForeColor, FillColor có giá trị miêu tả màu dạng RGB với qui định sau :

Color	Red Value	Green Value	Blue Value
Black	0	0	0
Blue	0	0	255
Green	0	255	0
Cyan	0	255	255
Red	255	0	0
Magenta	255	0	255
Yellow	255	255	0
White	255	255	255

- Thí dụ ta viết lệnh gán : Form1.BackColor = RGB(0,0,0) để thiết lập màu nền của form tên Form1 là màu đen.

Chi tiết về thuộc tính miêu tả màu (tt)

- Nếu chỉ muốn dùng 1 trong 16 màu cơ bản của QBasic (version Basic đầu tiên của Microsoft chạy trên DOS), ta có thể dùng hàm QBColor. Bảng sau liệt kê 16 màu cơ bản này :

Number	Color	Number	Color
0	Black	8	Gray
1	Blue	9	Light Blue
2	Green	10	Light Green
3	Cyan	11	Light Cyan
4	Red	12	Light Red
5	Magenta	13	Light Magenta
6	Yellow	14	Light Yellow
7	White	15	Bright White

- Thí dụ ta viết lệnh gán : Form1.BackColor = QBColor(15) để thiết lập màu nền của form tên Form1 là màu trắng sáng.

Chi tiết về thuộc tính mẫu tô nền

- Thuộc tính FillStyle có giá trị miêu tả 1 mẫu tô nền với qui định sau :

Constant	Setting	Description
VbFSSolid	0	Solid
VbFSTransparent	1	(Default) Transparent
VbHorizontalLine	2	Horizontal Line
VbVerticalLine	3	Vertical Line
VbUpwardDiagonal	4	Upward Diagonal
VbDownwardDiagonal	5	Downward Diagonal
VbCross	6	Cross
VbDiagonalCross	7	Diagonal Cross

- Thí dụ ta viết lệnh gán : Form1.FillStyle = VbVerticalLine để thiết lập mẫu tô nền của các phần tử trong form là các đường thẳng đứng.

Chi tiết về thuộc tính mẫu vẽ đường viền

- Thuộc tính DrawStyle có giá trị miêu tả 1 mẫu vẽ đường viền với qui định sau :

Constant	Setting	Description
VbSolid	0 ————	(Default) Solid
VbDash	1 — — —	Dash
VbDot	2 ----------	Dot
VbDashDot	3 - - — - —	Dash-Dot
VbDashDotDot	4 ---—---	Dash-Dot-Dot
VbInvisible	5	Transparent
VbInsideSolid	6 ————	Inside Solid

- Thí dụ ta viết lệnh gán : Form1.DrawStyle = VbDash để thiết lập mẫu vẽ đường viền của các phần tử trong form là các đường gạch-gạch dài.
- Lưu ý thuộc tính DrawStyle chỉ có nghĩa theo bảng trên khi ta thiết lập thuộc tính DrawWidth = 1. Trong trường hợp DrawWidth > 1 thì DrawStyle <> 5 đều tạo ra nét vẽ liên tục.

Function PaintPicture : vẽ ảnh bitmap bất kỳ

- Mỗi đối tượng vẽ có method PaintPicture cho phép ta vẽ ảnh bitmap bất kỳ. Cú pháp như sau :
  [*objName*.]**PaintPicture** *picture*, *dx, dy, dw, dh*, s*x*, s*y*, s*w*, s*h*, *opcode*
  trong đó :
  - *objName* là tên của Form, PictureBox hay Printer, nếu không có thì form hiện hành được vẽ.
  - *picture* là ảnh bitmap gốc được dùng để vẽ.
  - *dx, dy* là toạ độ đỉnh trên trái của vùng chứa ảnh vẽ trong đối tượng vẽ.
  - *dw, dh* là độ rộng, độ cao của vùng chứa ảnh vẽ trong đối tượng vẽ.
  - *sx, sy* là toạ độ đỉnh trên trái của vùng chứa ảnh trong ảnh gốc.
  - *sw, sh* là độ rộng, độ cao của vùng chứa ảnh trong ảnh gốc.
  - *opcode* miêu tả hành vi đưa ảnh gốc vào đối tượng vẽ, ta thường dùng các mã sau :
    vbSrcCopy : copy ảnh gốc vào vị trí qui định của đối tượng vẽ.
    vbSrcPaint : Or từng pixel ảnh gốc với từng bit đối tượng vẽ tương ứng.
    vbSrcInvert : Xor từng pixel ảnh gốc với từng bit đối tượng vẽ tương ứng.
    vbSrcAnd : And từng pixel ảnh gốc với từng bit đối tượng vẽ tương ứng...

Thí dụ về việc dùng function PaintPicture

Tạo 1 form trống, 'add' điều khiển PictureBox chứa ảnh gốc vào form, set thuộc tính Visible = False, thuộc tính Picture = đường dẫn file ảnh, rồi viết đoạn code sau cho form :

```
Option Explicit
Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
Private Sub Form_Load()
   AutoRedraw = False                              ' để VB gọi hàm xử lý biến cố Paint
End Sub

Private Sub Form_Paint()                           ' hiển thị ảnh chạy từ từ sang phải
Dim x As Integer, y As Integer
    ScaleMode = vbPixels                           ' đơn vị tính của form là pixel
    Picture1.ScaleMode = vbPixels                  ' đơn vị tính ở ảnh gốc là pixel
    x = 0                                          ' thiết lập vị trí đầu của ảnh
    y = 50
    While True
        PaintPicture Picture1, x, y, 60, 60, 0, 0, , , vbMergePaint        ' vẽ ảnh ở vị trí x,y
        DoEvents                                   ' cho phép ứng dụng đáp ứng sự kiện
        Sleep (10)                                 ' ngủ chờ 10ms
        Line (x, y)-(x + 60, y + 60), BackColor, BF   ' xóa ảnh vừa vẽ
        x = x + 4                                  ' di chuyển vị trí về bên phải 4 pixel
        If (x - 60 > ScaleWidth) Then x = 0        ' nếu ảnh đạt lề phải thì set về trái
    Wend
End Sub
```


10.5 Vấn đề in ấn trong VB

- Trong chương trình VB, ta có thể in thông tin ra máy in bằng cách dùng 1 trong 3 cách sau :
  - dùng các method vẽ văn bản, đồ họa và ảnh bitmap lên 1 form theo ý muốn rồi gọi method PrintForm để in form kết quả ra máy in. Đây là cách dễ dàng nhất để kiểm tra kết quả trước khi in ra giấy thực sự, nhưng kết quả có độ phân giải không cao (vì trùng với độ phân giải của màn hình).
  - dùng các method vẽ văn bản, đồ họa và ảnh bitmap theo ý muốn trực tiếp lên đối tượng Printer cùng 2 method điều khiển NewPage & EndDoc để xuất kết quả trực tiếp ra máy in default của Windows. Cách này cho kết quả có độ phân giải đúng với máy in (thường rất cao so với độ phân giải màn hình).
  - dùng lệnh **Set Printer = Printers(n)** để chọn máy in cụ thể trong danh sách các driver máy in hiện có của Windows rồi dùng các method vẽ văn bản, đồ họa và ảnh bitmap theo ý muốn trực tiếp lên đối tượng Printer cùng 2 method điều khiển NewPage & EndDoc để xuất kết quả trực tiếp ra máy in vừa chọn.

MÔN TIN HỌC

Chương 11

QUẢN LÝ HỆ THỐNG FILE

11.1 Tổng quát về truy xuất file trong VB
11.2 Qui trình điển hình để truy xuất Binary file
11.3 Qui trình điển hình để truy xuất Random file
11.4 Qui trình điển hình để truy xuất Sequential file
11.5 Các hàm truy xuất thuộc tính file
11.6 Các lệnh xử lý thư mục

Tổng quát về thời gian sống của biến dữ liệu

- Chương trình xử lý dữ liệu thông qua các biến dữ liệu. Như ta đã biết, mỗi biến dữ liệu chỉ có đời sống ngắn ngủi : hoặc bằng đời sống của 1 thủ tục, hoặc bằng đời sống của 1 module (hay đối tượng) hoặc cao nhất là bằng thời gian chạy ứng dụng, từ lúc ứng dụng được nạp vào bộ nhớ trong đến lúc chương trình kết thúc thực thi.
- Để lưu giữ giá trị của 1 số biến hầu trao đổi dữ liệu giữa 2 ứng dụng khác nhau hay giữa 2 lần chạy khác nhau của cùng 1 ứng dụng, ta sẽ ghi giá trị các biến này ra môi trường chứa tin bền vững trên những đơn vị chứa tin độc lập được gọi là file.
- Trong chương 2, chúng ta đã trình bày các khái niệm về file, cách tổ chức đĩa cứng thành cây phân cấp các file cũng như các thao tác quản lý hệ thống file trực tiếp bởi người dùng.
- Trong chương này ta sẽ nghiên cứu chi tiết các phương tiện mà VB cung cấp để thực hiện các thao tác quản lý hệ thống file, đặc biệt là việc truy xuất nội dung của file, từ trong code của ứng dụng VB.

Cấu trúc file

- Ở cấp độ HĐH, file là danh sách gồm n byte chưa có ngữ nghĩa.
- Chính ứng dụng phải tự qui định cấu trúc cụ thể của file mình tạo ra/đọc lại và ngữ nghĩa của từng đơn vị cấu trúc này.
- VB cung cấp cho ứng dụng 3 dạng file khác nhau, ứng với mỗi dạng file có 1 cách thức truy xuất dữ liệu tương ứng :
  - **file tuần tự** (Sequential File) hay file văn bản là danh sách gồm n byte, mỗi byte là 1 ký tự ANSI. Thí dụ file source code của các class VB *.bas.
  - **file nhị phân** (Binary File) là danh sách gồm n byte nhị phân chưa có cấu trúc. Thí dụ file Word, file Excel, file khả thi - executable.
  - **file truy xuất trực tiếp** (Random File) là danh sách gồm n record có cùng độ dài, mỗi record chứa nhiều field thông tin. Thí dụ file chứa các hồ sơ sinh viên.

Tổng quát về quản lý hệ thống file trong VB

- VB cung cấp cho người lập trình 2 phương pháp khác nhau để quản lý hệ thống file :
  - **gọi các thủ tục truyền thống** như Open, Close, Input, Write, Get, Put... Ta có thể gọi các thủ tục này là các lệnh VB.
  - **dùng mô hình đối tượng FSO** (File System Object). Với phương pháp này, đầu tiên người lập trình sẽ tạo ra đối tượng FileSystemObject rồi mỗi khi cần quản lý hệ thống file, họ chỉ cần gọi method tương ứng của đối tượng trên.
- Mặc dù FSO là phương pháp hướng đối tượng, rất thân thiện và dễ dùng, nhưng hiện FSO chưa đủ mạnh, chỉ cho phép truy xuất file text -văn bản, chứ chưa cho phép truy xuất 2 dạng file Binary và Random. Do đó trong chương này chúng ta chỉ tập trung giới thiệu phương pháp dùng các thủ tục truyền thống để truy xuất file. Sau này khi có điều kiện, mỗi SV sẽ tự nghiên cứu thêm cách dùng mô hình FSO.

11.1 Tổng quát về truy xuất file trong VB

- Qui trình truy xuất 1 file trong VB gồm 3 bước chính :
  - **mở/tạo file** (gọi thủ tục Open) : khai báo cho hệ thống biết ta sắp sửa truy xuất 1 file được xác định bởi đường dẫn cụ thể cùng các chế độ truy xuất file cụ thể.
  - **lặp truy xuất file** thông qua việc gọi các thủ tục Input, Get, Write, Put,... Thường mỗi thủ tục trên chỉ truy xuất 1 đơn vị thông tin nhỏ của file nên ta phải lặp nhiều lần đến khi hết file, tuy nhiên ta có quyền đọc/ghi toàn bộ nội dung file vào/ra bộ nhớ.
  - **đóng file** (gọi thủ tục Close) : khai báo cho hệ thống biết ta không còn muốn truy xuất file nữa để hệ thống cấm không cho bất kỳ lệnh nào trong ứng dụng của ta truy xuất file nữa.
- Để giúp các bạn dễ dàng tiếp thu thông tin, chúng tôi sẽ trình bày qui trình truy xuất file chi tiết trên từng loại file cụ thể : Text file, Random file và Binary File.

Bảng các lệnh VB truy xuất file

Tên phát biểu	Sequential File	Random File	Binary File
Open	x	x	x
Close	x	x	x
Input #	x		
Line Input #	x		
Print #	x		
Write #	x		
Input ()	x		x
Type .... EndType		x	
Put		x	x
Get		x	x

Bảng các function truy xuất file

- Dir : duyệt các phần tử trong 1 thư mục
- FileCopy : nhân bản vô tính 1 file thành file mới
- FileDateTime : đọc/hiệu chỉnh ngày/giờ hiệu chỉnh file lần cuối
- FileLen : xác định độ dài file chưa mở
- FreeFile : xác định chỉ số file còn trống để có thể dùng an toàn
- Loc : xác định vị trí truy xuất hiện hành trong file
- LOF : xác định độ dài file đã mở
- Seek : xác định/thiết lập vị trí truy xuất file
- GetAttr : đọc thuộc tính file
- SetAttr : ghi thuộc tính file

11.2 Qui trình điển hình để truy xuất Binary file

- Cú pháp lệnh open file nhị phân như sau :
  Open *pathname* **For Binary As** *filenumber*
- Nếu chưa biết cấu trúc dữ liệu của file, nên dùng biến dãy các byte để đọc/ghi dữ liệu từ/ra file nhị phân. Nếu đã biết cấu trúc dữ liệu của file, nên khai báo kiểu dữ liệu miêu tả cấu trúc đó rồi khai báo biến có kiểu vừa định nghĩa để chứa thông tin đọc từ file vào.
- Dùng lệnh **Seek [#]***filenumber*, *position* để dời pointer đọc/ghi tới vị trí xác định trước khi truy xuất.
- Dùng lệnh **Get** [**#**]*filenumber*, [*position*], *AVariable* để đọc dữ liệu từ vị trí *position* rồi chứa vào biến AVariable. Số byte được đọc = kích thước của biến dữ liệu.
- Dùng lệnh **Put** [**#**]*filenumber*, [*position*], *AVariable* để ghi nội dung của biến *AVariable* ra file từ vị trí *position.* Số byte được ghi = kích thước của biến dữ liệu.
- Sau khi đã xử lý xong file, ta dùng lệnh Close **[#]***filenum* để đóng file lại.

Thí dụ truy xuất Binary file

- Mỗi file khả thi (executable) đều có 1 header dài 20h (32) byte với các field như sau :

Index	Field	Diễn giải
0	Magic	Chuỗi 2 ký tự magic "MZ"
...	...	các field khác chưa cần chú ý
18h	Offset	Offset tới bảng tái định chương trình stub

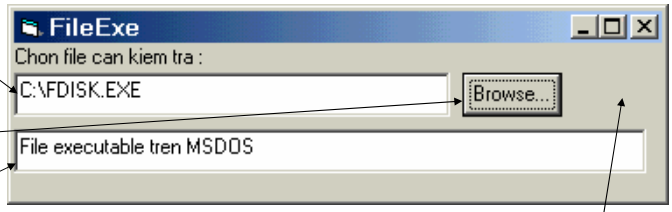
  Nếu giá trị field Offset (2 byte) ở offset 18h (24) có giá trị là 40h (64) thì file tương ứng là file executable trên Windows (*.exe, *.dll, *.ocx, *.scr, *.drv,...). Còn nếu giá trị Offset < 40h thì file tương ứng là file *.exe chạy trên DOS.
- Dựa vào Header trên, ta thử viết 1 ứng dụng cho phép user chọn đường dẫn của 1 file bất kỳ rồi kiểm tra tính chất của file đó và hiển thị kết quả :
  - Không phải file executable
  - File executable chạy trên DOS
  - File executable trên Windows (*.exe, *.dll, *.ocx, *.scr, *.drv,...).

Giao diện đề nghị của thí dụ truy xuất Binary file

Textbox hiển thị đường dẫn file cần khảo sát.

Button duyệt và chọn file.

Textbox chứa kết quả khảo sát.



CommonDialog duyệt file (bị ẩn)

```
Option Explicit
' Thủ tục xử lý sự kiện Click button
Private Sub cmdBrowse_Click()
    CommonDialog1.ShowOpen          ' hiển thị cửa sổ duyệt và chọn file
    txtFileName.Text = CommonDialog1.FileName
    CheckFileClass (txtFileName.Text)
End Sub
```

Chi tiết hàm kiểm tra loại file

```
Private Sub CheckFileClass(strFileName As String)
Dim FileNum As Integer
Dim strMagic As String * 2
Dim intOffset As Integer
    FileNum = FreeFile                  ' Tìm chỉ số file chưa dùng và mở file
    Open strFileName For Binary As FileNum
    Get FileNum, 1, strMagic            ' đọc 2 chuỗi 2 ký tự magic đầu file
    If strMagic <> "MZ" Then
        txtFileClass.Text = "Khong phai file executable"
        Exit Sub
    End If
    Get FileNum, &H19, intOffset        ' đọc word Offset tới chương trình Stub
    If intOffset < &H40 Then
        txtFileClass.Text = "File executable tren MSDOS"
    Else
        txtFileClass.Text = "File executable tren Windows (*.exe, *.dll, *.ocx,...)"
    End If
    Close #FileNum
End Sub
```


11.3 Qui trình điển hình để truy xuất Random file

- Cú pháp lệnh open file Random như sau :
  Open *pathname* [**For Random] As** *filenumber* Len = *reclength*
- Phải khai báo kiểu dữ liệu miêu tả cấu trúc của từng record dữ liệu được đọc/ghi rồi khai báo biến có kiểu vừa định nghĩa để chứa thông tin đọc từ /ghi ra file.
- Dùng lệnh **Seek [#]***filenumber*, *position* để dời pointer đọc/ghi tới vị trí record xác định.
- Dùng lệnh **Get** [**#]***filenumber*, [*position*], *AVariable* để đọc dữ liệu từ vị trí *position* rồi chứa vào biến AVariable. Số byte được đọc = kích thước của record dữ liệu.
- Dùng lệnh **Put** [**#]***filenumber*, [*position*], *AVariable* để ghi nội dung của biến *AVariable* ra file từ vị trí *position.* Số byte được ghi = kích thước của biến record dữ liệu.
- Sau khi đã xử lý xong file, ta dùng lệnh Close **[#]***filenumber* để đóng file lại.

Thí dụ truy xuất Random file

```
Type SVRecord
    hoten As String *30
    tuoi As Byte
    diachi As String *50
    lop As String*10
    ...
End Type
Dim MyRecord As Record   ' định nghĩa biến chứa từng record của file.
Dim MaxSize As Long, RecordNumber As Long
' 1. mở file ở chế độ random-file.
Open "HosoSinhvien" For Random As #1 Len = Len(MyRecord)
MaxSize = LOF(1) \ Len(MyRecord)   ' Tính số record trong file.
' 2. lặp đọc từng record từ cuối file lên đầu file
For RecordNumber = MaxSize To 1 Step - 1
    Seek #1, RecordNumber   ' thiết lập vị trí truy xuất.
    Get #1, , MyRecord   ' đọc record.
    ' Xử lý record vừa đọc vào
    ...
Next RecordNumber
' 3. đóng file.
Close #1
```


11.4 Qui trình điển hình để truy xuất Sequential file

- Cú pháp lệnh open file Sequential như sau :
  Open *pathname* [**Input | Output | Append] As** *filenumber* *[*Len = *buffersize]*
- Nếu file được mở ở chế độ Input & chưa tồn tại thì lỗi sai xãy ra. Nếu file được mở ở chế độ Output | Append & chưa tồn tại thì hệ thống sẽ tạo ra file mới.
- Dùng lệnh **Line Input #***filenumber*, *varname* để đọc 1 hàng văn bản từ file (kết thúc bởi ký tự CR - Carriage Return hay CRLF).
- Dùng lệnh **Input #***filenumber, varlist* để đọc các chuỗi hay số từ file (được trình bày chi tiết trong slide kế).
- Dùng hàm **Input(***number*, [**#**]*filenumber*) để đọc 1 chuỗi từ file (được trình bày chi tiết trong slide kế).
- Dùng lệnh **Write #***filenumber*, [*outputlist*] để ghi các chuỗi hay số ra file (được trình bày chi tiết trong slide kế).
- Sau khi đã xử lý xong file, ta dùng lệnh Close **[#]***filenum* để đóng file lại.

Chi tiết về lệnh Input

- Cú pháp : **Input #***filenumber, varlist*

  trong đó *varlist* là danh sách các biến chứa giá trị chuỗi hay số cần đọc, các biến trong danh sách được ngăn cách bằng dấu ','.
- Cách thức xử lý dữ liệu trên file nhập :

Dạng dữ liệu trên file	**Giá trị nhận được**
dấu ',' hay dòng trống	Empty
#NULL#	Null
#TRUE# hay #FALSE#	True hay False
#yyyy-mm-dd hh:mm:ss#	Ngày/giờ
#ERROR errornumber#	mã lỗi errornumber
"abcdef"	chuỗi abcdef
1254.386	giá trị 1254.386

Chi tiết về hàm Input

- Ngoài lệnh Input vừa giới thiệu ở slide trước, VB cung cấp thêm hàm Input với đặc tả sau : **Input** *number*, [**#**]*filenumber*

  trong đó *number* là số ký tự ANSI cần đọc từ file *filenumber*.
- Khác với lệnh Input, kết quả trả về của hàm Input gồm mọi ký tự thô trên file kể cả các ký tự điều khiển CR, LF, khoảng trắng, ",", nháy kép...
- Chỉ dùng hàm Input trên các file được mở ở chế độ **Input** l **Binary.**

Chi tiết về lệnh Write

- Cú pháp : **Write #***filenumber,* [*outputlist*]

  trong đó *outputlist* là danh sách các biểu thức chứa giá trị chuỗi hay số cần ghi, các biểu thức trong danh sách được ngăn cách bằng dấu ','.
- Cách thức xử lý in dữ liệu ra file :

Kiểu dữ liệu	**Kết quả được ghi trên file**
outputlist chỉ có dấu ','	dòng trống (CRLF)
Null	#NULL#
luận lý	#TRUE# hay #FALSE#
Date	#yyyy-mm-dd hh:mm:ss#
mã lỗi errornumber	#ERROR errornumber#
chuỗi abcdef	"abcdef"
số	chuỗi miêu tả số dùng dấu '.'

- Các kết quả in được ngăn cách với nhau bởi dấu ',' trên file xuất.

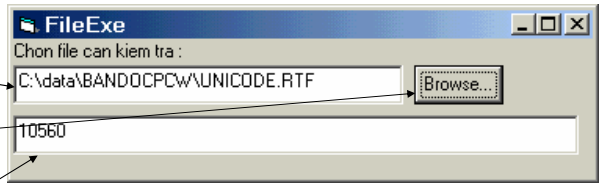
Thí dụ truy xuất Sequential file

- Ta thử viết ứng dụng cho phép user chọn đường dẫn của 1 file text-only rồi đếm số từ được chứa trong file này. Giả sử mỗi từ là 1 chuỗi ký tự chữ số (a-z hay 0-9) bất kỳ được ngăn cách nhau bởi các ký tự không phải là chữ số.

Textbox hiển thị đường dẫn file cần khảo sát.

Button duyệt và chọn file.

Textbox chứa số từ trong file.




```
Option Explicit
' Thủ tục xử lý sự kiện Click button Browse
Private Sub cmdBrowse_Click()
   CommonDialog1.ShowOpen
   txtFileName.Text = CommonDialog1.FileName
   txtWordCount.Text = WordCount(txtFileName.Text)
End Sub
```

Chi tiết hàm đếm từ trong file

```
Private Function WordCount(strFileName As String) As Long
Dim lngWcount As Long, FileNum As Integer, Char As String * 1
    FileNum = FreeFile                  ' Tìm chỉ số file chưa dùng và mở file
    Open strFileName For Input As FileNum
    lngWcount = 0
    Do While Not EOF(FileNum)           ' Lặp đến hết file.
        Do                              ' tìm các dấu ngăn trước 1 từ mới
            Char = Input(1, #1)
            Char = LCase(Char)
        Loop Until ("a" <= Char And Char <= "z") Or ("0" <= Char And Char <= "9")_
                Or EOF(FileNum)
        If EOF(FileNum) Then GoTo CloseRet
        lngWcount = lngWcount + 1   ' tăng số từ lên 1
        Do                          ' tìm và bỏ các ký tự của từ hiện hành
            Char = Input(1, #1)
            Char = LCase(Char)
        Loop Until Not (("a" <= Char And Char <= "z") Or ("0" <= Char And Char <= "9"))_
                Or EOF(FileNum)
        If EOF(FileNum) Then GoTo CloseRet
    Loop
CloseRet:
    WordCount = lngWcount
    Close #FileNum
End Function
```


Các hàm xác định vị trí truy xuất trong file

- Hàm **Loc** (*filenum*) trả về vị trí truy xuất hiện tại trong file *filenum*.

Mode	Return Value
Random	chỉ số record đọc/ghi lần cuối cùng.
Binary	chỉ số byte đọc/ghi lần cuối cùng.
Output	chỉ số byte đọc/ghi lần cuối cùng /128
Append	(nhưng thường không dùng kết quả này)
Input	

Ví dụ :

```
Dim MyChar As Byte
Open "TestFile" For Binary As #1   ' mở file để đọc/ghi.
Do While Not EOF(1)   ' lặp cho đến hết file.
        MyChar = Input(1, #1)   ' đọc byte kế tiếp.
        Debug.Print Loc(1)   ' hiển thị vị trí byte vừa được đọc
Loop
Close #1   ' đóng file.
```

Các hàm xác định vị trí truy xuất trong file (tt)

- Hàm **Seek** (*filenum*) trả về vị trí truy xuất kế tiếp trong file *filenum*.

Mode	Return Value
Random	hoạt động truy xuất kế tiếp sẽ xảy ra ở vị trí record này
Binary	hoạt động truy xuất kế tiếp sẽ xảy ra ở vị trí byte này
Output, **Append**, **Input**	vị trí byte đầu tiên là 1, kế tiếp là 2...

Ví dụ :
```
Dim MyChar As Byte
Open "TestFile" For Input As #1   ' mở file để đọc.
Do While Not EOF(1)   ' lặp cho đến hết file.
        MyChar = Input(1, #1)   ' đọc ký tự (byte) kế tiếp.
        Debug.Print Seek(1)   ' hiển thị vị trí byte sẽ đọc kế tiếp
Loop
Close #1   ' đóng file.
```


Các hàm xác định vị trí truy xuất trong file (tt)

- Lệnh **Seek** [**#**]*filenumber*, *position* thiết lập vị trí truy xuất kế tiếp trong file *filenum*.

Mode	Return Value
Random	hoạt động truy xuất kế tiếp sẽ xảy ra ở vị trí record *position*
Binary	hoạt động truy xuất kế tiếp sẽ xảy ra ở vị trí byte *position*
Output, **Append**, **Input**	vị trí byte đầu tiên là 1, kế tiếp là 2...

11.5 Các hàm truy xuất thuộc tính file

- Hàm **FileDateTime** (*pathname*) trả về ngày/giờ hiệu chỉnh lần cuối của file xác định bởi thông số *pathname*.

  Ví dụ : Dim MyStamp As Date
  MyStamp = FileDateTime ("c:\windows\win.com")

- Hàm **GetAttr** (*pathname*) trả về byte thuộc tính của file xác định bởi thông số *pathname*. Thứ tự các bit thuộc tính trong byte thuộc tính như sau :

Giá trị	Tên hằng gợi nhớ	Diễn giải
0	vbNormal	Normal
1	vbReadOnly	file chỉ đọc
2	vbHidden	file ẩn
4	vbSystem	file hệ thống
8	vbVolume	tên nhãn đĩa
16	vbDirectory	thư mục
32	vbArchive	file bị thay đổi từ lần backup cuối

Các hàm truy xuất thuộc tính file (tt)

Ví dụ :
```
If GetAttr ("c:\windows\win.com") And vbReadOnly Then
   MsgBox "c:\windows\win.com là file chỉ đọc"
End If
```

- Hàm **SetAttr** *pathname, attributes* thiết lập thuộc tính của file xác định bởi thông số *pathname* theo byte *attributes.*

  Ví dụ : đoạn code sau sẽ thiết lập thuộc tính của file c:\windows\win.com về read-only.

```
Dim bytFileAttr As Byte
bytFileAttr = GetAttr ("c:\windows\win.com")
bytFileAttr = bytFileAttr Or vbReadOnly          ' Or bitwise
SetAttr "c:\windows\win.com", bytFileAttr
```

Các hàm truy xuất thuộc tính file (tt)

- Hàm **FreeFile [***(rangenumber)***]** trả về 1 số thuộc kiểu **Integer** miêu tả chỉ số file chưa được dùng (để ta dùng an toàn). Thường không cần dùng tham số khi gọi hàm này.
- Hàm **LOF** (*#filenum*) trả về 1 giá trị thuộc kiểu **Long** miêu tả kích thước của file đã được mở và hiện được xác định bởi thông số *#filenum*.

  Ví dụ :
```
Dim lngFileSize As Long, filenum As Integer
filenum = FreeFile
Open "c:\windows\win.com" For Input As #filenum      ' mở file
lngFileSize = LOF (filenum)      ' lấy kích thước file đã mở #1
...
Close #1                         ' đóng file
```
- Hàm **FileLen** (*pathname*) trả về 1 giá trị thuộc kiểu **Long** miêu tả kích thước của file xác định bởi thông số *pathname*.

  Ví dụ :
```
Dim lngFileSize As Long
lngFileSize = FileLen ("c:\windows\win.com")
```


Lệnh nhân bản và xóa file

- Lệnh **FileCopy** *SourcePath*, *DestPath*.
  nhân bản file *SourcePath* ra file *DestPath.* Không được mở file rồi nhân bản nó.
  Ví dụ : FileCopy c:\autoexec.bat c:\backup\autoexec.bat
- Lệnh **Kill** *pattern*
  xóa từ 0 đến n file có tên thỏa mãn mẫu *pattern* (dùng ký tự * và ? để miêu tả).
  Ví dụ : Kill "c:\windows*.tmp"

11.6 Các lệnh xử lý thư mục

- Lệnh **MkDir** *pathname* : tạo thư mục mới.
  Ví dụ : MkDir "c:\windows\temp"
- Lệnh **RmDir** *pathname* : xóa thư mục trống có tên là *pathname*, nếu thư mục chưa trống thì phải dùng lệnh Kill và/hoặc RmDir để xóa các file và thư mục con của nó trước.
  Ví dụ : RmDir "c:\windows\tmp"
- Hàm **CurDir** *[(drive)]* : trả về đường dẫn của thư mục làm việc của đĩa hiện hành (hoặc của đĩa được xác định bởi thanh số).
  Ví dụ : MyPath = CurDir ' trả về "c:\windows" nếu ta đang làm việc ở đây.
- Lệnh **ChDir** *pathname* : chuyển thư mục làm việc của ổ đĩa hiện hành về vị trí qui định bởi tham số.
  Ví dụ : ChDir "c:\windows\tmp"
- Lệnh **ChDrive** *drive* : chuyển ổ đĩa hiện hành về ổ đĩa được xác định trong tham số.
  Ví dụ : ChDrive "e"

Hàm duyệt các phần tử trong 1 thư mục

- Cú pháp : **Dir** [**(***pattern*[**,** *attributes*]**)**]
  trong đó *pattern* xác định mẫu các phần tử cần tìm, *attributes* miêu tả byte thuộc tính của phần tử cần tìm.
  Nếu có tham số *pattern* thì hàm Dir sẽ trả về đường dẫn của phần tử đầu tiên tìm được (hay chuỗi rỗng nếu không tìm được), nếu không có *pattern* thì hàm Dir trả về đường dẫn của phần tử kế tiếp thỏa pattern được xác định lần cuối.
  Ý nghĩa các bit trong byte attributes đã được trình bày trong silde 261.

Thí dụ xóa file đệ qui

- Trong lúc hoạt động, các ứng dụng thường tạo ra các file tạm có phần nới rộng là *.tmp, *.bak,... Khi kết thúc, ứng dụng sẽ xóa các file tạm đi. Tuy nhiên trong 1 số trường hợp đặc biệt (máy treo, mất điện,...) các file tạm không được xóa hết và vẫn tồn tại trên đĩa cứng ở nhiều thư mục khác nhau. Ta hãy thử viết 1 ứng dụng cho phép user xác định mẫu các file cần xóa rồi tìm các file thỏa mãn mẫu qui định và xóa chúng, việc tìm và xóa nên đệ qui từ vị trí mẫu để xóa triệt để. Thí dụ nếu người dùng nhập pattern c:*.tmp, ứng dụng sẽ tìm và xóa mọi file *.tmp từ thư mục gốc.

Giao diện đề nghị của ứng dụng

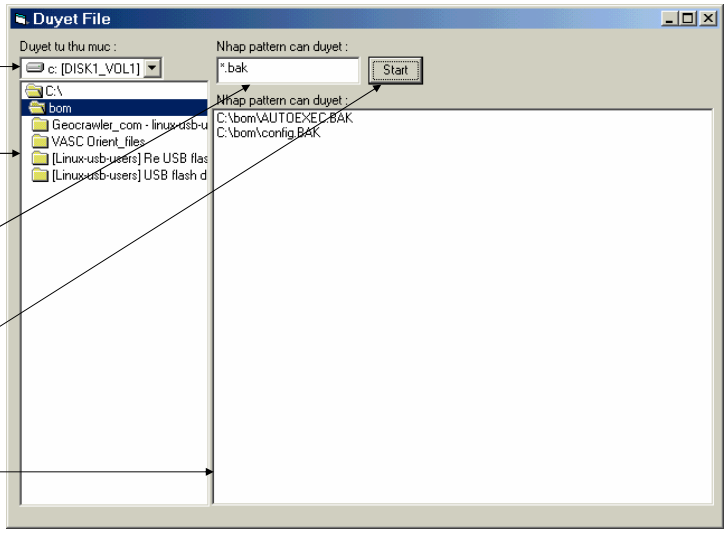
DriveListBox chọn ổ đĩa.

DirListBox chọn thư mục.

TextBox nhập pattern file cần xóa.

Button bắt đầu duyệt file đệ qui và xóa.

ListBox hiển thị các file bị xóa.



Code của ứng dụng xóa file đệ qui

```
Const QMAX = 10000
Dim strDirQueue(0 To QMAX) As String    ' Hàng chứa các thư mục cần duyệt
Dim iHead As Integer                    ' chỉ số trong hàng chứa thư mục sắp duyệt
Dim iTail As Integer                    ' chỉ số trong hàng sắp chứa thư mục cần duyệt

Private Sub Form_Resize()
   ScaleMode = vbPixels
   Dir1.Move 8, 45, 170, Me.ScaleHeight - 50
   FileList.Move 180, 70, Me.ScaleWidth - 185, Me.ScaleHeight - 75
End Sub

Private Sub Drive1_Change()             ' thủ tục xử lý sự kiện chọn ổ đĩa
    Dir1.Path = Drive1.Drive
End Sub

Private Sub Dir1_Change()               ' thủ tục xử lý sự kiện chọn thư mục
    If (Right(Dir1.Path, 1) = "\") Then
        strDirQueue(0) = Dir1.Path
    Else
        strDirQueue(0) = Dir1.Path & "\"
    End If
End Sub
```


Code của ứng dụng xóa file đệ qui (tt)

```
' Thủ tục xử lý sự kiện Click button Start
Private Sub cmdStart_Click()
Dim bytPredIdx As Byte
Dim bytCurIdx As Byte
Dim strTmp As String
    bytPredIdx = 0
    bytCurIdx = 1
    iHead = 0
    iTail = 1
    While iHead < iTail
        Call DuyetXoaFileDequi
        iHead = iHead + 1
        If (iHead > QMAX) Then iHead = 0
    Wend
End Sub
```

Code của ứng dụng xóa file đệ qui (tt)

```
Private Sub DuyetXoaFileDequi()
Dim Name As String, Path As String, strFilePath As String, intAttr As Integer
   Path = strDirQueue(iHead)
   Name = Dir(Path, vbDirectory)                ' lấy 1 phần tử thỏa mãn pattern
   Do While Name <> ""                          ' Lặp xử lý phần tử, nếu còn.
      If Name <> "." And Name <> ".." Then      ' bỏ thư mục hiện hành và cha của nó
         strFilePath = Path & Name
         intAttr = GetAttr(strFilePath)
         If (intAttr And vbDirectory) = vbDirectory Then      ' nếu là thư mục thì lưu giữ vào hàng
            strDirQueue(iTail) = strFilePath & "\"
            iTail = iTail + 1
            If (iTail > QMAX) Then iTail = 0
         Else                                   ' nếu là file kiểm tra xem thỏa pattern không
            If LCase(Name) Like txtPattern.Text Then
               intAttr = intAttr And Not vbReadOnly
               SetAttr strFilePath, intAttr
               Kill strFilePath
               FileList.AddItem strFilePath
            End If
         End If
      End If
      Name = Dir   ' lấy phần tử kế thỏa mãn pattern.
   Loop
End Sub
```


MÔN TIN HỌC

Chương 12

LINH KIỆN PHẦN MỀM & TRUY XUẤT DATABASE

12.1 Tổng quát về linh kiện phần mềm
12.2 Cách tạo và dùng linh kiện phần mềm
12.3 Tổng quát về truy xuất database
12.4 Tổng quát về debug mã nguồn VB

12.1 Tổng quát về linh kiện phần mềm

- Mục tiêu của qui trình phát triển phần mềm hướng đối tượng là tạo ra ứng dụng có cấu trúc thuần nhất : tập các đối tượng sống và tương tác lẫn nhau.
- Mỗi đối tượng của ứng dụng có thể được tạo ra nhờ 1 trong các cách sau :
  - chọn menu Project.Add Class Module để tạo ra 1 class module mới miêu tả cấu trúc chi tiết của đối tượng cần tạo ra : các thuộc tính dữ liệu và các method của đối tượng.
  - chọn menu Project.Add File rồi khai báo đường dẫn của file *.bas chứa 1 class module của 1 ứng dụng có sẵn để copy class module này vào Project ứng dụng hiện tại (đây là 1 phương pháp để thừa kế thành quả).
  - sử dụng các điều khiển sẵn có của VB để xây dựng form giao diện.
  - 'add' module *.ocx chứa 1 hay nhiều ActiveX Control đang được Windows quản lý vào Project ứng dụng hiện tại để dùng chúng y như các điều khiển sẵn có của VB.

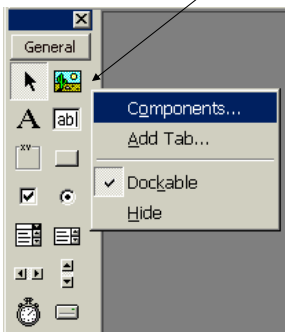
12.2 Cách tạo và dùng linh kiện phần mềm

- VB cho phép tạo linh kiện phần mềm ActiveX Control nhờ 1 trong 3 loại Project ActiveX EXE, ActiveX DLL và ActiveX Control. Tuy nhiên qui trình chi tiết để tạo ActiveX Control vượt quá khuôn khổ nội dung của môn học này.
- Việc dùng ActiveX Control cũng giống như dùng control định sẵn của VB, ta đặt chúng 1 cách trực quan vào các form giao diện với kích thước và vị trí phù hợp với nhu cầu. Khi viết code, ta có thể truy xuất các thuộc tính dữ liệu và các method của ActiveX Control y như truy xuất các thành phần trong control định sẵn.

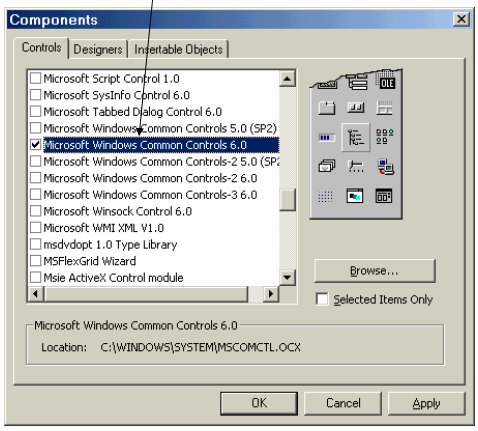
Qui trình 'add' 1 ActiveX Control vào Project

Để dùng 1 linh kiện phần mềm ActiveX Control trong 1 form của Project ứng dụng, trước hết ta phải thêm nó vào cửa sổ Toolbox của Project theo qui trình điển hình sau đây :

1. ấn phải chuột vào vị trí trống của Toolbox, chọn mục Components



2. chọn tab Controls, duyệt và chọn mục tương ứng, chọn OK.



3. Cửa sổ Toolbox sẽ chứa các icon miêu tả các Act. Control trong module vừa chọn.

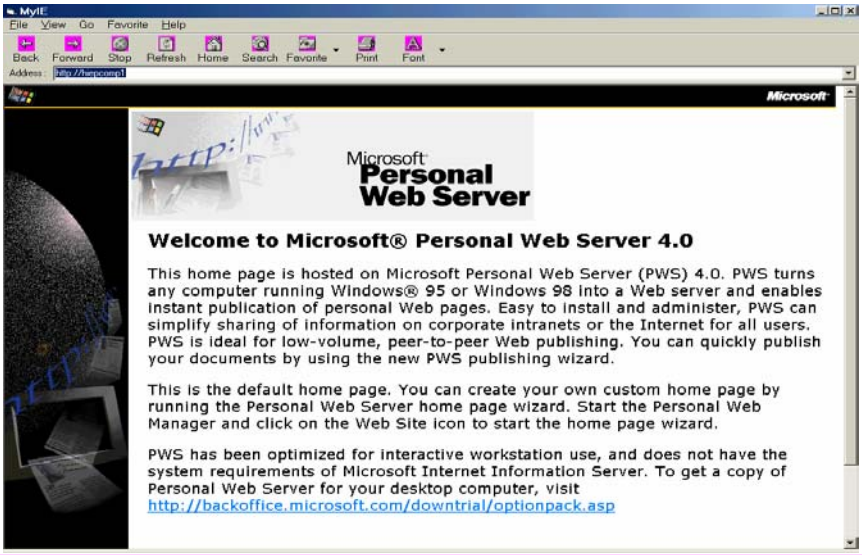
Thí dụ về cách dùng ActiveX Control

- Để thấy việc dùng ActiveX Control hầu xây dựng phần mềm dễ dàng như thế nào, chúng ta hãy thử viết 1 trình duyệt Web với chức năng tượng tự như IE của Microsoft, ta tạm gọi ứng dụng sắp viết này là MyIE.
- Việc viết phần mềm duyệt Web từ đầu rất khó khăn vì bạn cần phải trang bị nhiều kiến thức như : kỹ thuật hiển thị văn bản và đồ họa, kỹ thuật tương tác với người dùng thông qua bàn phím và chuột, kỹ thuật và qui trình viết 1 chương trình dịch, lập trình mạng dùng socket, giao thức truy xuất tài nguyên Web HTTP (Hypertext Transfer Protocol), ngôn ngữ DHTML,...
- Nhưng toàn bộ các công việc mà 1 trình duyệt Web cần làm đã được Microsoft đóng gói trong 1 linh kiện phần mềm tên là WebBrowser.

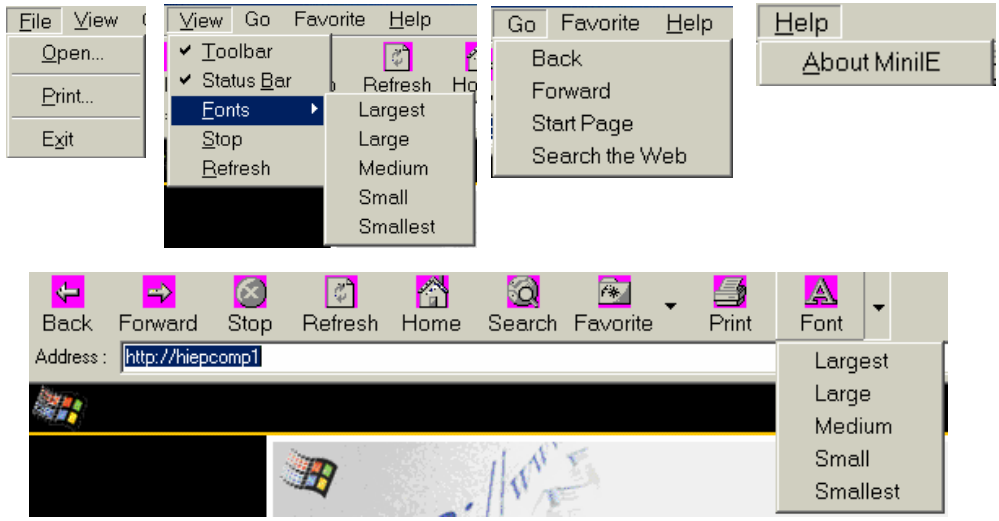
Thí dụ về cách dùng ActiveX Control (tt)

- ActiveX Control "WebBrowser" là 1 đối tượng giao diện chỉ chứa 1 vùng hiển thị nội dung hình chữ nhật với vị trí và kích thước do người lập trình qui định. Interface của nó bao gồm 3 loại : các thuộc tính interface (khoảng 27), các method (khoảng 12) và các sự kiện (khoảng 16) mà người dùng có thể lập trình thủ tục đáp ứng cho chúng. Ở đây chúng ta sẽ giới thiệu 1 số method mà ta dùng trong việc viết ứng dụng MyIE :
  - **Navigate2** (URL,...) cho phép download trang Web hay file *.doc, *.xls, *.ppt,... được xác định bởi URL, hiển thị nội dung của nó lên vùng hiển thị rồi chờ và xử lý sự tương tác của người dùng (ấn vào vùng liên kết,...).
  - **GoBack** cho phép quay về trang Web vừa truy xuất (ngay trước trang hiện hành).
  - **GoForward** cho phép tiến tới trang Web (ngay sau trang hiện hành).
  - **GoHome** cho phép hiển thị trang chủ.
  - **Refresh** cho phép download và hiển thị lại trang hiện hành.
  - **Stop** cho phép dừng ngay việc download và hiển thị trang Web hiện hành.
  - **ExecWB** cho phép thực thi 1 số hành vi trên trang web như thay đổi co chữ, in trang Web ra máy in,...

Giao diện đề nghị của trình MyIE



Chi tiết các option trong menu và toolbar (tt)



Phân tích & thiết kế giao diện

- Trình MyIE có giao diện SDI gồm 1 menu bar, 1 toolbar, 1 ComboBox liệt kê các URL vừa truy xuất, 1 ActiveX Control "WebBrowser" xử lý việc truy xuất, hiển thị các trang Web và chờ phục vụ các tương tác của người dùng. Qui trình chi tiết xây dựng ứng dụng MyIE sẽ được trình bày trong bài thực hành số 4, ở đây chỉ tóm tắt các bước chính :
  - Tạo project loại "VB Application Wizard" và trả lời các bước Wizard để tạo Project.
  - Chọn Tools.Menu Editor để tạo menu bar theo đặc tả của slide trước. Qui trình dùng Menu Editor được giới thiệu trong chương 4.
  - Cũng đọc lại chương 4 để biết qui trình tạo/hiệu chỉnh Toolbar.
  - tạo (vẽ) ComboBox nhập URL mới và liệt kê các URL đã truy xuất.
  - tạo (vẽ) control "WebBrowser". Không cần vẽ ComboBox và WebBrowser chính xác vì code sẽ xác định động mỗi lần cửa sổ chương trình thay đổi.
  - tạo thủ tục xử lý sự kiện cho các menu option và toolbar button rồi viết code cho chúng. Code của các thủ tục này chủ yếu làm "cò" và gọi các method tương ứng trong đối tượng WebBrowser thực thi.

Phân tích & thiết kế giao diện (tt)

- Lưu ý rằng trước khi thiết kế được giao diện của trình MyIE, bạn cần 'add' các linh kiện ActiveX Control sau đây :
  - Microsoft Common Dialog Control 6.0.
  - Microsoft Internet Control.
  - Microsoft Windows Common Controls 6.0.
- Lưu ý rằng qui trình Wizard cho loại Project SDI đã tạo sẵn cho ta 1 form của chương trình tên là frmMain. Form này đã chứa sẵn 1 menu bar, 1 Toolbar. Chúng ta chỉ cần hiệu chỉnh lại 2 thành phần này chứ không cần phải tạo mới chúng.

12.3 Tổng quát về truy xuất database

- Trong chương 11, chúng ta đã giới thiệu qui trình truy xuất dữ liệu trên các file. Một trong các dạng file đã giới thiệu là Random File, file này là danh sách các record dữ liệu có cấu trúc và độ dài giống nhau, mỗi record chứa nhiều field dữ liệu, thí dụ file chứa các hồ sơ sinh viên, file chứa các hồ sơ nhà, file chứa các hồ sơ đường xá...
- Hầu hết các ứng dụng hiện nay (nhất là các ứng dụng nghiệp vụ) đều phải truy xuất các random file. Việc quản lý các random file bao gồm nhiều tác vụ như tạo file mới với cấu trúc record cụ thể, thêm/bớt/hiệu chỉnh/duyệt các record, tìm kiếm các record thỏa mãn 1 tiêu chuẩn nào đó,... Để thực hiện các tác vụ trên (nhất là tìm kiếm record) hiệu quả, tin cậy, ta cần nhiều kiến thức khác nhau và phải tốn nhiều công sức.
- Random file (với 1 số sự cải tiến và tăng cường) được gọi là database quan hệ. Có nhiều format database quan hệ khác nhau đang được dùng. Để giải phóng các ứng dụng khỏi việc quản lý database, người ta đã xây dựng ứng dụng đặc biệt : DBMS (Database Management System). Ứng dụng sẽ nhờ DBMS để truy xuất database được dễ dàng.

Các giao tiếp lập trình truy xuất database

- Về nguyên tắc, ứng dụng VB (hay viết bằng ngôn ngữ khác) có thể truy xuất 1 database bằng 1 trong các giao tiếp lập trình sau đây :

ADO (ActiveX Data Objects)
DAO (Data Access Objects)
ODBC (Open Database Connectivity)
DBMS-Specific Language
Direct to database (file)

- Trong các giao tiếp truy xuất database trên chỉ có ADO là dễ dàng hơn cả, đại đa số trường hợp ta chỉ tạo trực quan các ActiveX Control và khai báo các thuộc tính của chúng là đã truy xuất được database mà không cần viết code. Trong trường hợp phải viết code thì cũng rất ngắn và dễ dàng.

Truy xuất database dùng ADO thông qua các ActiveX Control

- Truy xuất database dùng ADO thông qua các ActiveX Control là phương pháp truy xuất database trực quan và dễ dàng nhất.
- Đa số các database trên Windows do Access tạo ra trong đó mỗi file database chứa nhiều table, mỗi table là danh sách n record có cấu trúc chung nào đó. Qui trình điển hình truy xuất các record của 1 table trong 1 database Access có thể dùng các đối tượng sau :
  - Dùng đối tượng ADODB trong thư viện "Microsoft ActiveX Data Objects 2.x Library" để có thể liệt kê các table trong 1 database Access.
  - Dùng đối tượng Microsoft Data Control 6.0 để quản lý 1 RecordSet chứa tập các record của 1 table thỏa mãn 1 điều kiện nào đó.
  - Dùng đối tượng Microsoft DataGrid Control 6.0 để hiển thị các record của 1 Data Control và cho phép user thêm/bớt/hiệu chỉnh các record.
- Để thấy rõ việc truy xuất database là rất dễ dàng, ta hãy viết 1 ứng dụng truy xuất database dạng Access có giao diện như slide sau :

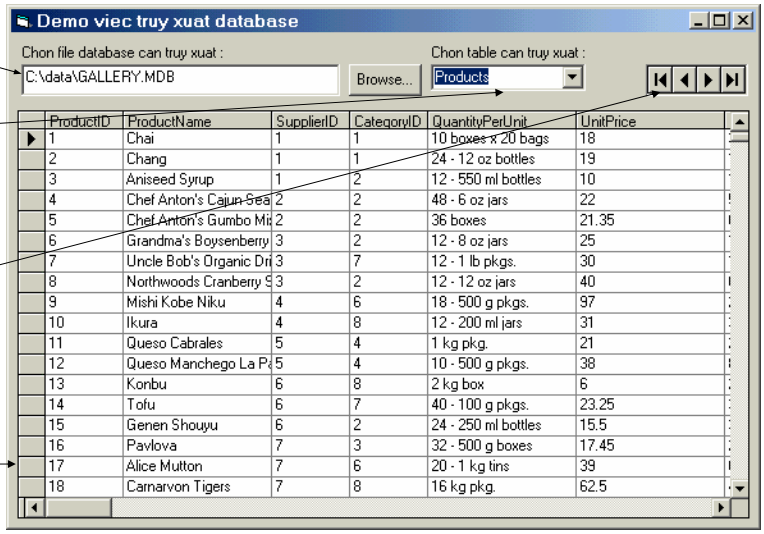
Giao diện đề nghị của ứng dụng truy xuất database

TextEdit qui định file cần truy xuất.

ComboBox liệt kê các Table trong file.

ADODC quản lý các record thỏa mãn 1 điều kiện mong muốn (có thể ẩn đi).

DataGrid hiển thị các record trong ADODC để user khảo sát và hiệu chỉnh.



ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice
1	Chai	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Sea	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mi	2	2	36 boxes	21.35
6	Grandma's Boysenberry	3	2	12 - 8 oz jars	25
7	Uncle Bob's Organic Dr	3	7	12 - 1 lb pkgs.	30
8	Northwoods Cranberry S	3	2	12 - 12 oz jars	40
9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	97
10	Ikura	4	8	12 - 200 ml jars	31
11	Queso Cabrales	5	4	1 kg pkg.	21
12	Queso Manchego La P	5	4	10 - 500 g pkgs.	38
13	Konbu	6	8	2 kg box	6
14	Tofu	6	7	40 - 100 g pkgs.	23.25
15	Genen Shouyu	6	2	24 - 250 ml bottles	15.5
16	Pavlova	7	3	32 - 500 g boxes	17.45
17	Alice Mutton	7	6	20 - 1 kg tins	39
18	Carnarvon Tigers	7	8	16 kg pkg.	62.5

Qui trình xây dựng ứng dụng của slide trước

- Ta có thể tạo project dạng "Standard EXE", để dùng các điều khiển trong Form ta cần 'add' các ActiveX Control sau vào Project :
  - Microosft Data Control 6.0 để quản lý 1 RecordSet chứa tập các record trong 1 table.
  - Microsoft DataGrid Control 6.0 để hiển thị các record của 1 Data Control và cho phép user thêm/bớt/hiệu chỉnh các record.
  - Microsoft Common Dialog Control 6.0 để hiển thị cửa sổ duyệt file và chọn file cần truy xuất.
- Để dùng được đối tượng ADODB trong Project, ta chọn menu Project.References để chọn thư viện sau :
  - Microsoft ActiveX Data Objects 2.x Library, với x =1 | 2 | 3 | 4 | 5 ...
- Thiết kế trực quan form theo slide trước, tạo các thủ tục xử lý sự kiện cho button Browse và sự kiện Click cho ComboBox.

Chi tiết các thủ tục xử lý sự kiện

```
' Thủ tục xử lý click button Browse
Private Sub cmdBrowse_Click()
    ' hiển thị dialog box duyệt và chọn file
    CommonDialog1.ShowOpen
    ' hiển thị file được chọn vào textbox
    txtFileName.Text = CommonDialog1.FileName
    ' duyệt các table và hiển thị tên của chúng vào ComboBox
    ListAccessTables (txtFileName.Text)
End Sub

' Thủ tục xử lý khi user chọn Table trong danh sách
Private Sub cbTable_Click()
   Adodc1.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;_
        Data Source=" & txtFileName.Text
   Adodc1.RecordSource = cbTable.Text
   Adodc1.Refresh
End Sub

' Thủ tục phục vụ sự kiện form bị thay đổi kích thước.
Private Sub Form_Resize()
    ' vẽ lại DataGrid để phù hợp với kích thước mới của form
    ScaleMode = vbPixels
    RsList.Move 5, 60, Me.ScaleWidth - 10, Me.ScaleHeight - 65
End Sub
```


Chi tiết thủ tục hiển thị danh sách các Table của database

```
Private Sub ListAccessTables(strDBPath As String)
Dim adoConnection As ADODB.Connection, adoRsFields As ADODB.Recordset
    While cbTable.ListCount <> 0                ' Xóa danh sách hiện hành
        cbTable.RemoveItem 0
    Wend
    ' Tạo 1 connection đến file database
    Set adoConnection = New ADODB.Connection
    adoConnection.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & strDBPath
    ' Duyệt các tables, 'add' tên của từng table vào danh sách ComboBox.
    Set adoRsFields = adoConnection.OpenSchema(adSchemaTables)
    With adoRsFields
        Do While Not .EOF
            If .Fields("TABLE_TYPE") = "TABLE" Then
                cbTable.AddItem .Fields("TABLE_NAME")
            End If
            .MoveNext
        Loop
    End With
    adoRsFields.Close               ' đóng và xóa recordset chứa các table
    Set adoRsFields = Nothing
    adoConnection.Close             ' đóng và xóa connection đến file database
    Set adoConnection = Nothing
End Sub
```

Lập trình truy xuất database dùng ADO

- Định nghĩa DSN (Data source name) miêu tả file database, nếu có bước này, ứng dụng truy xuất database thông qua tên luận lý DSN mà không cần biết chính xác vị trí đường dẫn file database và máy chứa file database. Dùng icon "ODBC ..." trong Control Panel của Windows để định nghĩa DSN.
- Trong ứng dụng ta dùng đối tượng ADODB để truy xuất database theo qui trình điển hình sau :
  1. tạo 1 **đối tượng Connection** miêu tả database cần truy xuất.
  2. gọi **method OpenSchema** trên đối tượng Connection để tìm các Table trên database.
  3. khi user chọn 1 Table (hay dùng lệnh SQL để miêu tả điều kiện xác định các record), ta gọi **method Execute** trên đối tượng Connection để tạo 1 **đối tượng RecordSet** chứa các record tìm được.
  4. duyệt các record trong RecordSet và xử lý chúng theo yêu cầu.
  5. đóng và xóa RecordSet và lập lại các bước 3, 4 để xử lý Table khác.
  6. đóng và xóa các đối tượng đã tạo ra để giải phóng bộ nhớ do chúng chiếm.

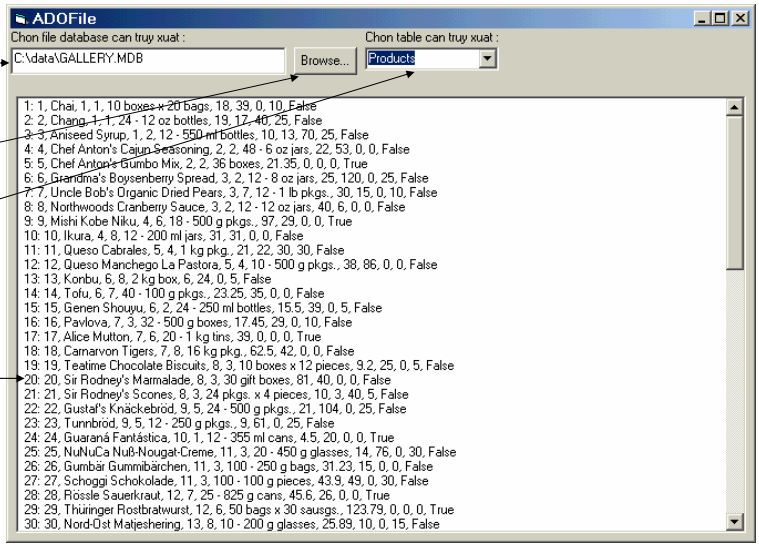
Thí dụ lập trình truy xuất database dùng ADO

TextEdit qui định file cần truy xuất.

Button duyệt và chọn file database.

ComboBox liệt kê các Table trong file.

Listbox hiển thị các record trong 1 Table đã chọn (để xem chứ không hiệu chỉnh).



Chi tiết các thủ tục xử lý sự kiện

```
Dim adoConnection As ADODB.Connection          ' biến tham khảo đến Connection
Dim adoRsFields As ADODB.Recordset             ' biến tham khảo đến RecordSet

' Thủ tục xử lý click button Browse
Private Sub cmdBrowse_Click()
    ' hiển thị dialog box duyệt và chọn file
    CommonDialog1.ShowOpen
    ' hiển thị file được chọn vào textbox
    txtFileName.Text = CommonDialog1.FileName
    ' duyệt các table và hiển thị tên của chúng vào ComboBox
    ListAccessTables (txtFileName.Text)
End Sub

' Thủ tục phục vụ sự kiện form thay bị thay đổi kích thước.
Private Sub Form_Resize()
    ' vẽ lại ListBox để phù hợp với kích thước mới của form
    ScaleMode = vbPixels
    RsList.Move 5, 60, Me.ScaleWidth - 10, Me.ScaleHeight - 65
End Sub
```


Chi tiết thủ tục hiển thị danh sách các Table của database

```
Private Sub ListAccessTables(strDBPath As String)
    While cbTable.ListCount <> 0              ' Xóa danh sách hiện hành
        cbTable.RemoveItem 0
    Wend
    ' 1. Tạo 1 connection đến file database
    Set adoConnection = New ADODB.Connection
    adoConnection.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & strDBPath
    ' 2. Duyệt các tables, 'add' tên của từng table vào danh sách ComboBox.
    Set adoRsFields = adoConnection.OpenSchema(adSchemaTables)
    With adoRsFields
        Do While Not .EOF
            If .Fields("TABLE_TYPE") = "TABLE" Then
                cbTable.AddItem .Fields("TABLE_NAME")
            End If
            .MoveNext
        Loop
    End With
    adoRsFields.Close                 ' đóng và xóa recordset chứa các table
    Set adoRsFields = Nothing
    ' lưu ý vẫn để mở Connection đến file database hầu truy xuất lại
End Sub
```

Chi tiết các thủ tục xử lý sự kiện

```
' Thủ tục xử lý click chọn table trong ComboBox
Private Sub cbTable_Click()
Dim fcount As Integer, rcount As Integer, strBuf As String
    ' 3. Tạo đối tượng RecordSet chứa các record của Table được chọn
    Set adoRsFields = adoConnection.Execute("SELECT * FROM " & cbTable.Text)
    ' 4. Duyệt các record trong RecordSet và hiển thị chúng trong ListBox
    With adoRsFields
        rcount = 0                      ' chỉ số record
        fcount = .Fields.Count          ' số field trong từng record
        Do While Not .EOF               ' duyệt từng record trong recordset
            rcount = rcount + 1
            strBuf = Str(rcount) & ": " & .Fields(0).Value
            For idx = 1 To fcount - 1   ' duyệt từng field và nối kết vào chuỗi strBuf
                strBuf = strBuf & ", " & .Fields(idx).Value
            Next
            RsList.AddItem strBuf       ' add chuỗi kết quả vào Lisbox
            .MoveNext                   ' di chuyển đến record kế tiếp
        Loop
    End With
    adoRsFields.Close                   ' đóng và xóa recordset
    Set adoRsFields = Nothing
End Sub
```


12.4 Tổng quát về hoạt động debug ứng dụng

- Sau khi viết code cho ứng dụng xong, ta sẽ thử chạy nó để xác định xem nó giải quyết đúng yêu cầu không. Thường ứng dụng chứa nhiều lỗi sai thuộc 1 trong 2 loại sau :
  - các lỗi về từ vựng (tên các phần tử, từ dành riêng,..) và cú pháp của các phần tử cấu thành ứng dụng. VB sẽ phát hiện các lỗi này dễ dàng và hiển thị thông báo lỗi cho ta xem xét và sửa chữa. Thường sau khi được VB thông báo về các lỗi này, ta dễ dàng sửa chúng.
  - các lỗi về giải thuật của ứng dụng. VB không thể phát hiện các lỗi này vì chúng thuộc phạm trù ngữ nghĩa. Ứng dụng sẽ chạy theo giải thuật được miêu tả, ta phải tự đánh giá tính đúng/sai về giải thuật, nhưng việc tìm lỗi giải thuật thường rất khó. Để giúp đỡ người lập trình dễ dàng tìm ra các lỗi giải thuật, VB cung cấp công cụ cho phép họ kiểm soát được qui trình chạy ứng dụng và truy xuất các biến dữ liệu của chương trình, công cụ này được gọi là 'Debug'.

Tổng quát về hoạt động debug ứng dụng

Trong quá trình debug, ứng dụng sẽ ở 1 trong 2 trạng thái sau :

- **Pause** : trạng thái của ứng dụng trước khi chạy hay khi dừng lại theo 1 điều kiện dừng nào đó của người debug. VB sẽ ghi nhớ lệnh sắp thi hành trước khi dừng (lệnh đầu tiên của ứng dụng nếu nó chưa bắt đầu chạy). Do tính lịch sử, ta dùng thuật ngữ PC - program counter để nói về lệnh này. Ở trạng thái này, người debug có thể xem giá trị của các biến dữ liệu để biết ứng dụng chạy đúng hay sai theo yêu cầu rồi điều khiển việc thi hành tiếp theo của ứng dụng, lúc này ứng dụng sẽ chuyển sang trạng thái Running.
- **Running** : trạng thái mà ứng dụng đang chạy các lệnh của nó đến khi nó gặp 1 điều kiện dừng đã thiết lập trước, lúc này ứng dụng sẽ chuyển về trạng thái **Pause**.

Trong quá trình debug, ứng dụng ở trạng thái Pause chủ yếu thời gian và người debug tương tác với ứng dụng chủ yếu ở trạng thái này. Mỗi khi ứng dụng được chạy tiếp, nó chuyển qua trạng thái Running, nhưng sẽ nhanh chóng chạy đến lệnh dừng và chuyển về trạng thái Pause (trừ phi bị 'block' chờ I/O hay bị 'loop' trong các vòng lặp vô tận).

Các thao tác để xem và hiệu chỉnh biến dữ liệu

Để xem nội dung của 1 biến dữ liệu, người debug có thể :

- chọn menu Debug.Add Watch để thêm 1 biểu thức (thường là biến dữ liệu) vào cửa sổ Watch để xem nội dung của nó.
- chọn menu Debug.Edit Watch để hiệu chỉnh biểu thức (thường là biến dữ liệu) hiện hành ở cửa sổ Watch (context, watch type).
- dời chuột đến tên biến trong cửa sổ code, 1 cửa sổ nhỏ chứa giá trị của biến đó sẽ được hiển thị để người debug xem xét.

Để hiển thị cửa sổ chứa danh sách các thủ tục đang thực hiện dỡ dang (các thủ tục lồng nhau theo thứ tự), người debug có thể :

- chọn menu View.Call Stacks.

Để xem vị trí PC hiện hành (lệnh sắp thực hiện kế tiếp), người debug có thể :

- chọn menu Debug.Show Next Statement (thường khi ứng dụng dừng lại, nó sẽ hiển thị lệnh chạy kế tiếp - lệnh bị dừng với màu tô đặc biệt và có dấu mũi tên ở lề trái của lệnh).
- chọn menu Debug.Set Next Statement để thiết lập lệnh chứa cursor hiện hành là lệnh chạy kế tiếp (thay vì lệnh bị dừng trước đó)

Các lệnh thiết lập điều kiện dừng

Nếu điều kiện dừng là vị trí lệnh cụ thể thì người debug có thể :

- chọn menu Debug.Clear All Breakpoints để xóa tất cả các điểm dừng (breakpoint) hiện tại. Điểm dừng là lệnh mà khi ứng dụng chạy đến, ứng dụng sẽ dừng lại và chuyển về trạng tháo Pause để người debug kiểm soát nội dung dữ liệu của ứng dụng.
- chọn menu Debug.Toggle Breakpoint để thiết lập/xóa điểm dừng ở lệnh chứa cursor hiện hành (có thể thực hiện nhanh chức năng này bằng cách dời cursor đến lề trái của lệnh cần thiết lập/xóa điểm dừng rồi click chuột).

Nếu điều kiện dừng là biến dữ liệu/biểu thức nào đó bị thay đổi hay có giá trị True thì người debug có thể :

- chọn menu Debug.Add Watch, nhập biểu thức (thường là 1 biến dữ liệu) rồi chọn điều kiện dừng "Break when value is True" hay "Break when value changes".
- chọn menu Debug.Edit Watch, hiệu chỉnh biểu thức hiện hành (thường là 1 biến dữ liệu) rồi chọn điều kiện dừng "Break when value is True" hay "Break when value changes".

Ta có thể (và nên) thiết lập nhiều điều kiện dừng đồng thời để 'rào chắn' luồng thi hành của chương trình.

Các lệnh điều khiển chạy tiếp ứng dụng

Để chạy tiếp ứng dụng từ vị trí PC hiện hành, người debug có thể :

- chọn menu Run.Start để bắt đầu chạy ứng dụng, ứng dụng chỉ dừng lại khi gặp điều kiện dừng nào đó đã được thiết lập.
- chọn menu Run.Continue để chạy tiếp từ vị trí PC hiện hành, ứng dụng chỉ dừng lại khi gặp điều kiện dừng nào đó đã được thiết lập.
- chọn menu Debug.Step Over để chạy tiếp 1 lệnh rồi dừng lại (Pause), nếu lệnh thi hành là lệnh gọi thủ tục thì toàn bộ thủ tục sẽ được chạy. Đây là lệnh cho phép thực hiện từng lệnh theo mức vĩ mô.
- chọn menu Debug.Step Into để chạy tiếp 1 lệnh rồi dừng lại (Pause), nếu lệnh thi hành là lệnh gọi thủ tục thì ứng dụng sẽ dừng lại ở lệnh đầu tiên của thủ tục. Đây là lệnh cho phép thực hiện từng lệnh theo mức vi mô.
- chọn menu Debug.Step Out để chạy tiếp các lệnh còn lại của thủ tục hiện hành rồi quay về và dừng lại sau lệnh gọi thủ tục này (Pause).
- chọn menu Debug.Run to Cursor để chạy tiếp ứng dụng từ vị trí PC hiện hành đến lệnh chứa cursor hiện hành rồi dừng lại (Pause).

Các lệnh điều khiển khác

Khi ứng dụng ở trạng thái 'Pause', người debug có thể thực hiện các lệnh sau :

- chọn menu Run.End để kết thúc việc chạy ứng dụng.
- chọn menu Run.Restart để kết thúc việc chạy ứng dụng rồi bắt đầu chạy lại từ đầu.
- chọn menu Run.Break để dừng đột ngột việc chạy ứng dụng, lệnh đang thực hiện sẽ được đánh dấu để ta dễ theo dõi. Chức năng này giúp ta biết ứng dụng đang bị 'loop' ở đoạn lệnh nào. Nếu ứng dụng đang bị 'block' chờ biến cố I/O, sẽ không có lệnh nào được dánh dấu cả.