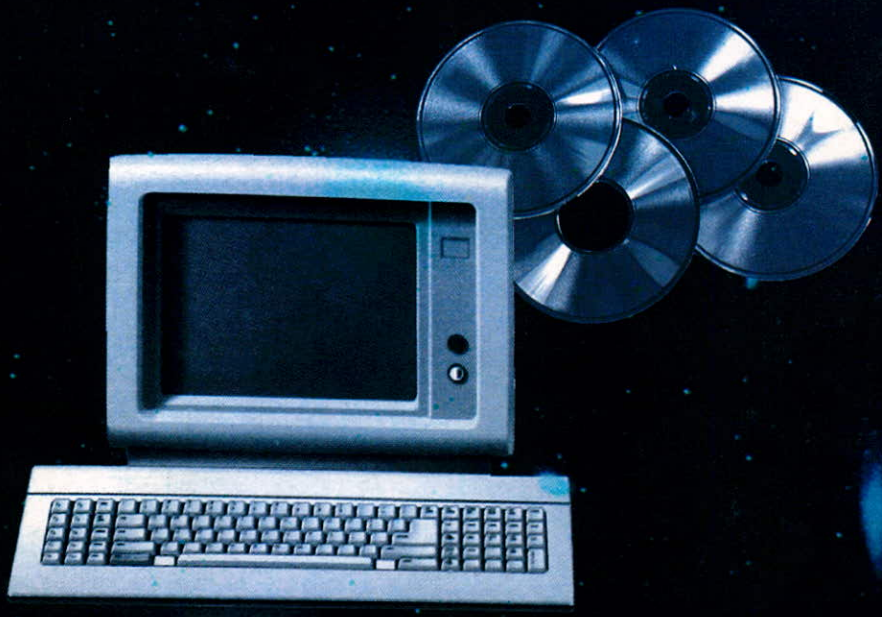
GS PHẠM VĂN ẤT

LẬP TRÌNH WINDOWS

Dùng ngôn ngữ C và các hàm API của Windows

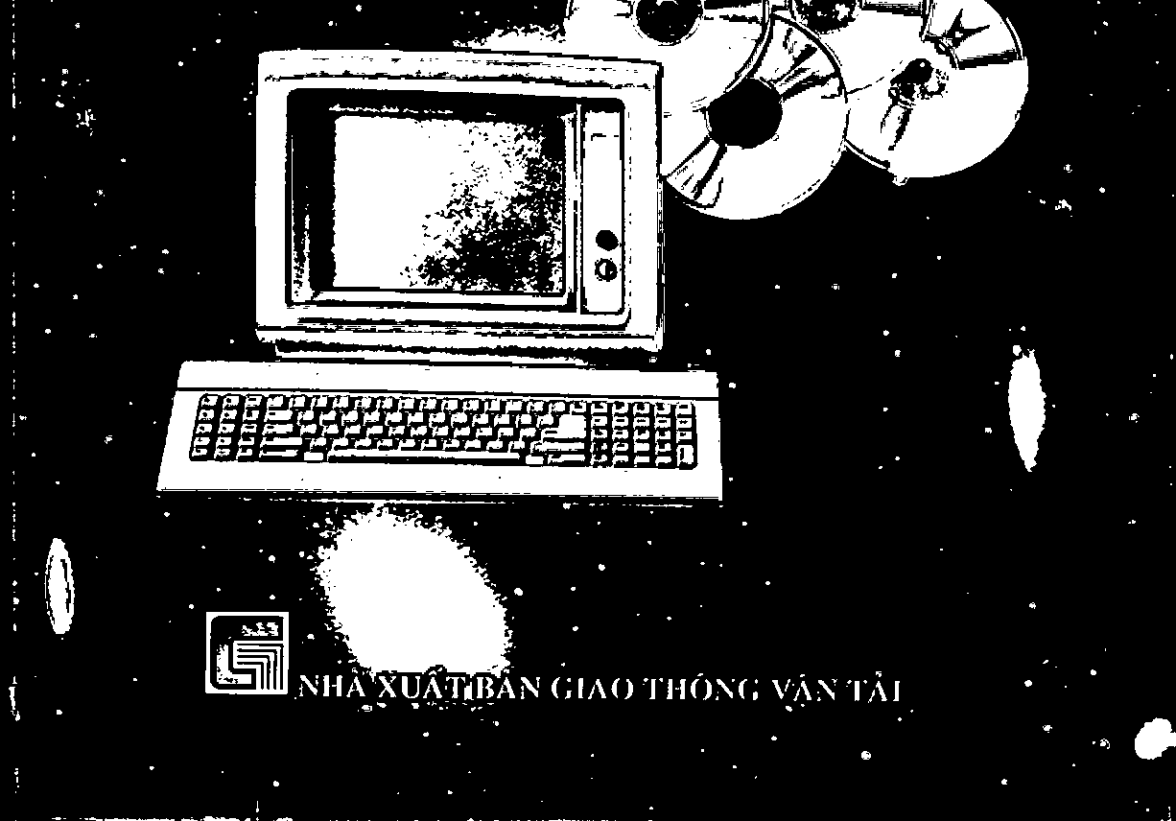


NHÀ XUẤT BẢN GIAO THÔNG VẬN TẢI

GS PHẠM VĂN ẤT

LẬP TRÌNH WINDOWS

Dùng ngôn ngữ C và các hàm API của Windows



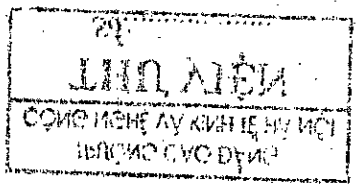
NHÀ XUẤT BẢN GIAO THÔNG VẬN TẢI

GS. PHẠM VĂN ẤT

LẬP TRÌNH WINDOWS

DÙNG NGÔN NGỮ C VÀ CÁC HÀM API CỦA WINDOWS

NHÀ XUẤT BẢN GIAO THÔNG VẬN TẢI



LỜI NÓI ĐẦU

Vào những năm 1970-1980 phần lớn các phần mềm ứng dụng được viết trên hệ điều hành MS DOS bằng các ngôn ngữ quen biết như Foxpro, Pascal và Turbo C. Nhưng từ khi hệ điều hành Windows 16 bit ra đời năm 1985, nhất là sau sự xuất hiện của các hệ điều hành Windows 32 bit như Windows 95, Windows 98 với các thế mạnh như giao diện đồ hoạ, tính đa nhiệm, khả năng quản lý bộ nhớ lớn, tính độc lập thiết bị và có nhiều công cụ, tài nguyên trợ giúp, thì các lập trình viên đã từ lập trình trên DOS chuyển sang trên Windows. Hiện nay phần các chương trình ứng dụng được chạy trên hệ điều hành Windows. Vì vậy môn học Lập trình Windows là rất cần thiết đối với sinh viên chuyên tin các trường đại học, cao đẳng cũng như đối với tất cả những người yêu thích tin học. Hiện tại môn học này đã được giảng dậy cho chuyên ngành tin học ở hầu hết các trường đại học và cao đẳng. Tuy nhiên so với lập trình trên DOS thì lập trình trên Windows phức tạp hơn và số tài liệu, giáo trình về vấn đề này chưa nhiều, chưa được thống nhất.

Bài giảng về Kỹ thuật lập trình Windows sẽ trình bầy cách thiết kế, cài đặt chương trình dựa trên thư viện các hàm API của Windows 32. Ngôn ngữ sử dụng là C trên Windows và môi trường thử nghiệm chương trình là Visual C++ 6.0 (phiên bản mới nhất hiện nay). Bài giảng được hình thành qua nhiều năm giảng dậy của tác giả tại nhiều trường đại học. Nội dung Bài giảng gồm 11 chương như sau.

Chương 1 "Khái quát về lập trình trên Windows" sẽ giới thiệu các ưu điểm nổi bật của chương trình chạy trên Windows so với chương trình chạy trên DOS như tính đa nhiệm, môi trường giao diện đồ hoạ, khả năng độc lập thiết bị và khả năng quản lý bộ nhớ lớn. Sau đó nói về cách tổ chức một chương trình, các tệp nguồn của chương trình, ví dụ về một chương trình. Để người đọc có thể thực hành trên máy, cuối chương trình bầy cách biên soạn, biên dịch và chạy một chương trình trong môi trường Visual C++ 6.0 và cách sử dụng các công cụ của Visual C++ 6.0 để tạo menu và hộp hội thoại - hai giao diện quan trọng của chương trình.

Chương 2 "Hàm WinMain và cửa sổ" trình bầy hai hàm quan trọng không thể thiếu của một chương trình C for Windows là hàm WinMain (có vai trò gần như hàm main của C for DOS) và hàm cửa sổ. Hàm WinMain là điểm khởi đầu và kết thúc chương trình, gồm các nội dung như: Đăng ký lớp cửa sổ, xây dựng và hiển thị cửa sổ chính của chương trình và tạo vòng lặp đợi, nhận, gửi thông điệp. Hàm cửa sổ có nhiệm vụ tiếp nhận và xử lý các thông điệp có liên quan đến cửa sổ, như thông điệp về việc bấm chuột tại một nút lệnh trên menu, các thông điệp liên quan đến sự thay đổi trạng thái của cửa sổ. Ngoài ra còn giới thiệu các hàm dùng để xây dựng và quản lý cửa sổ chính và cửa sổ con soạn thảo.

Chương 3 "Menu" sẽ giới thiệu cách tạo menu và gắn nó vào cửa sổ chính của chương trình. Trình bầy việc xử lý sự kiện chọn nút lệnh của menu, các

hàm dùng để quản lý trạng thái menu như đánh dấu một mục menu, làm một mục trở nên xám (gray), trở nên không hoạt động (disable) hay hoạt động (enable). Ngoài ra còn cho một danh sách các hàm liên quan đến menu.

Chương 4 "Hộp hội thoại" sẽ trình bầy các khái niệm về hộp hội thoại (giao diện vào ra quan trọng trong lập trình Windows), như thiết kế, đóng mở hộp thoại, các ô điều khiển trên hộp thoại. Ngoài ra sẽ giới thiệu 2 kiểu hộp thoại là modal và modeless

Chương 5 "Xử lý thông điệp" sẽ hệ thống hoá các loại thông điệp mà chương trình cần xử lý, như thông điệp menu, thông điệp liên quan đến sự thay đổi trạng thái của cửa sổ, thông điệp bàn phím, thông điệp chuột và thông điệp thời gian. Vì chương trình Windows hoạt động theo hướng sự kiện, nên việc hiểu biết các sự kiện và các thông điệp liên quan là rất cần thiết.

Chương 6 " Hiển thị văn bản lên màn hình" sẽ trình bầy hàm TextOut và các vấn đề liên quan dùng để in văn bản trên vùng làm việc của cửa sổ. Khác với màn hình văn bản trên DOS được tổ chức thành 80 cột và 25 dòng, màn hình trên Windows chỉ gồm các chấm điểm (pixel) nên người lập trình phải tự tính toán để tổ chức các dòng văn bản trên màn hình.

Chương 7 "Đồ hoạ" trình bầy các hàm dùng để vẽ các đường và các hình. Cũng sẽ giới thiệu các công cụ bổ trợ như bút vẽ, chổi tô, hộp mầu, đơn vị đo và hệ toạ độ.

Chương 8 "Sử dụng máy in" trình bầy các kiểu dữ liệu, các hàm và các bước tiến hành để in một văn bản từ cửa sổ con soạn thảo ra máy in. Cũng sẽ giới thiệu cách in các hình đồ hoạ.

Chương 9 "Thao tác tệp và cấp phát bộ nhớ" sẽ giới thiệu các hàm của Windows dùng để thao tác tệp và cấp phát bộ nhớ. Một điều cần nói là: Các hàm của C chuẩn về tệp và bộ nhớ vẫn sử dụng được trong C for DOS, nhưng việc sử dụng các hàm của Windows cho phép làm việc với các tệp và các khối bộ nhớ lớn hơn.

Chương 10 "Thư viện liên kết động" trình bầy cách xây dựng và sử dụng các tệp thư viện DLL mới. Đây là một công cụ tiện lợi trong việc tổ chức và phát triển chương trình.

Chương 11 "Ảnh bitmap" sẽ trình bầy cách hiển thị và in ảnh bitmap theo 2 cách. Cách thứ nhất rất đơn giản nhưng không linh hoạt vì cần biết tên và thư mục của tệp ảnh trong khi viết chương trình. Cách thứ hai linh hoạt và tổng quát hơn, cho phép dùng hộp thoại OpenFile để chọn một tệp bitmap từ một thư mục tuỳ ý. Để đạt được điều này, người lập trình phải biết được cấu trúc của tệp bitmap và biết cách đọc dữ liệu từ tệp vào bộ nhớ. Cuối chương trình bầy một chương trình với các chức năng hiển thị và in ảnh từ một tệp được chọn trên hộp thoại OpenFile.

Khi viết chúng tôi đã cố gắng để Bài giảng được hoàn chỉnh, nhưng chắc chắn không tránh khỏi thiếu sót. Rất mong nhận được sự đóng góp của bạn đọc để Bài giảng ngày một hoàn thiện hơn.

Tháng 05 năm 2004
Tác giả

CHƯƠNG 1

KHÁI QUÁT VỀ LẬP TRÌNH TRÊN WINDOWS

§1. ĐẶC ĐIỂM CỦA HỆ ĐIỀU HÀNH WINDOWS

1.1. Đây là hệ điều hành đa nhiệm (multi tasking). Cùng một lúc Windows quản lý và cho phép nhiều chương trình đồng thời thực hiện. Do đó các chương trình chạy trên Windows có những tính chất sau:

+ Các chương trình cần chia sẻ tài nguyên cho nhau (bộ nhớ, màn hình, bàn phím,...). Windows không cho phép một chương trình được độc chiếm bộ nhớ cũng như các thiết bị khác.

+ Chương trình không được trực tiếp nhận các tín hiệu nhập dữ liệu từ các thiết bị vào như bàn phím, chuột. Windows sẽ nhận các tín hiệu này và mã hoá thành các thông điệp dưới dạng số nguyên và gửi cho các ứng dụng (chương trình).

+ Chương trình không được sử dụng các hàm vào ra của C như các hàm scanf, printf mà phải sử dụng các hàm của Windows để tổ chức nhập xuất dữ liệu (trừ các hàm quản lý tệp tin).

+ Vì nhiều chương trình có thể đồng thời chạy trên Windows, nên có thể lấy kết quả chương trình này dùng cho chương trình khác. Cách đơn gian nhất để thực hiện điều này là dùng Clipboard của Windows .

+ Bản thân một chương trình có thể tái xuất hiện nhiều lần trong môi trương Windows. Điều này có thể xẩy ra khi một chương trình đang làm việc ta lại dùng chức năng Run để thực hiện nó hoặc dùng chuột kích hoạt biểu tượng của chương trình. Khi đó trên màn hình sẽ xuất hiện 2 cửa sổ hoàn toàn giống nhau của một chương trình. Nếu đóng một cửa sổ thì vẫn còn một cửa sổ khác. Từ đây nẩy sinh khái niệm "bản sao chương trình" (Instance). Khi chương trình chạy lần đầu ta được bản sao đầu tiên. Khi chương trình xuất hiện (chạy) lần thứ 2 ta được bản sao thứ 2,....

1.2. Quản lý sự kiện

+ Các tín hiệu nhập dữ liệu từ bàn phím, chuột được Windows nhận, mã hoá thành các thông điệp (Message) và chuyển vào hàng đợi của Windows. Sau đó Windows phân phối các thông điệp cho các ứng dụng. Mỗi ứng dụng

có một hàng đợi riêng (hàng đợi ứng dụng) để nhận các thông điệp do Windows gửi tới.

+ Một chương trình Windows cần có các hàm để tiếp nhận, phân loại và xử lý các thông điệp do Windows gửi tới. Đó là các hàm đặc biệt (không gặp trong C for DOS) lập ra để Windows gọi, chứ không được gọi trong chương trình (gọi là hàm Call back).

+ Các sự kiện có thể xảy ra: Chuột, bàn phím, thời gian, ...

1.3. Môi trường giao diện

+ Trong DOS màn hình được tổ chức thành 2 chế độ là đồ hoạ và văn bản. Trong Windows chỉ có một chế độ hiển thị màn hình là đồ hoạ, do đó việc xuất đồng thời dữ liệu đồ hoạ và văn bản ra màn hình được thực hiện khá dễ dàng nhờ các hàm API (Application Prorgam Interface) của Windows.

+ Trong Windows có sẵn nhiều công cụ giúp việc tổ chức giao diện chương trình rất thuận tiện như: cửa sổ, menu, hộp hội thoại, biểu tượng, con trỏ, ảnh Bitmap,... (gọi là các tài nguyên)

1.4. Khả năng độc lập thiết bị

+ Các hàm Windows cho phép tổ chức xuất dữ liệu trên các thiết bị khác nhau theo cùng một mẫu. Vì vậy khi thay đổi thiết bị in, thì chương trình chỉ phải sửa lại rất ít, còn các phần cơ bản thì vẫn giữ nguyên.

+ Cách tổ chức xuất dữ liệu theo 2 bước như sau:

1. Lấy chỉ danh ngữ cảnh thiết bị xuất.

2. Dùng các hàm của Windows để xuất dữ liệu ra thiết bị thông qua chỉ danh thiết bị.

Khi thay đổi thiết bị xuất ta chỉ cần sửa phần 1 để lấy chỉ danh thiết bị khác (chỉ một câu lệnh), còn phần 2 là phần chính thì giữ nguyên.

1.5. Khả năng quản lý bộ nhớ lớn

+ Trong DOS chỉ quản lý thuận lợi được vùng nhớ quy ước 640 KB. Vì vậy mặc dù máy có bộ nhớ đến hàng GB, các chương trình trong DOS vẫn thường thông báo không đủ bộ nhớ.

+ Windows quản lý được các khối bộ nhớ lớn hơn nhiều. Các hệ điều hành từ Windows 95 là các hệ điều hành 32 bit quản lý địa chỉ trực tiếp (không cần Segment và Offset như trong DOS) và có thể quản được bộ nhớ tới 4 GB.

+ Để tối ưu trong việc dùng bộ nhớ, Windows có thể cấp phát cho chương trình các khối nhớ có thể di chuyển, chứ không phải là các khối nhớ cố định. Do đó khi viết chương trình, cần chú ý đến đặc điểm này để truy xuất đến đúng vùng nhớ được cấp phát.

§2. TỔ CHỨC MỘT CHƯƠNG TRÌNH TRÊN WINDOWS

2.1. Giao diện

+ Mỗi chương trình giao tiếp với người sử dụng thông qua một cửa sổ chính (trong DOS mỗi chương trình độc chiếm màn hình). Hai thành phần quan trọng của cửa sổ là: một hệ menu để thể hiện các chức năng chính của chương trình và một vùng làm việc hình chữ nhật.

+ Trên vùng làm có thể hiển thị dữ liệu văn bản và đồ hoạ nhờ các hàm API của Windows. Chú ý là: không thể nhập dữ liệu trên vùng làm việc và không thể đặt con trỏ văn bản vào vùng làm việc. Tuy nhiên có thể đặt con trỏ chuột tại bất kỳ điểm nào trên vùng này.

+ Để nhập dữ liệu cần dùng các hộp hội thoại hoặc cửa sổ con soạn thảo.

2.2. Cấu trúc của chương trình windows

+ Hầu hết các chương trình windows được tổ chức thành các hàm sau :

- Hàm có tên WinMain, hàm này có vai trò như hàm main trong C for DOS và là hàm bắt buộc phải có trong chương trình.

- Hàm cửa sổ - Window (tên có thể đặt tuỳ ý nhưng thường dùng tên WndProc) dùng để đón nhận, phân loại và xử lý các sự kiện xảy ra trên cửa sổ chính (trên hệ menu và vùng làm việc của cửa sổ).

- Mỗi hộp hội thoại cần có một hàm DLG (Dialog) để tiếp nhận và xử lý các sự kiện trên hộp hội thoại.

2.3. Sử dụng các hàm chuẩn trong chương trình

+ Ngoài các hàm của C chuẩn (như sprintf, atof, memset,...), trong chương trình còn dùng các hàm API của Windows.

+ Thường dùng các hàm API để tổ chức giao diện nhập xuất dữ liệu, cấp phát bộ nhớ và dùng các hàm C để xử lý dữ liệu.

2.4. Các kiểu dữ liệu dùng trong chương trình

Trong chương trình C for Win có rất nhiều kiểu mới so với chương trình C for DOS, có thể phân chúng thành 4 nhóm sau:

2.4.1. Các kiểu dữ liệu dùng để mô tả, quản lý, sử dụng các đối tượng, khái niệm mới (chưa hề gặp trong DOS) như: bản sao chương trình, cửa sổ, menu, hộp thoại, ngữ cảnh thiết bị, ... Các đối tượng này được quản lý thông qua một kiểu gọi là: Chỉ danh (Handle). Một số kiểu thường gặp như:

HANDLE - Chỉ danh chung, thường dùng để khai báo và làm việc với bản sao chương trình.

HINSTANCE - Chỉ danh bản sao chương trình.

HWND - Chỉ danh cửa sổ, dùng để khai báo và làm việc với cửa sổ, hộp hội thoại, các ô điều khiển (control).

HMENU - Chỉ danh menu dùng để khai báo và làm việc với menu, menu con.

HDC - Chỉ danh ngữ cảnh thiết bị, dùng để làm việc với các thiết bị xuất như:

Vùng làm việc của cửa sổ (màn hình), máy in.

2.4.2. Các kiểu cấu trúc mới dùng để mô tả và tạo lập các đối tượng trong Windows như:

WNDCLASS - Chứa các thuộc tính của lớp cửa sổ, dùng để đăng ký một lớp cửa sổ mới.

MSG - Chứa thông tin về một thông điệp (Message).

RECT - Chứa thông tin về vùng làm việc của cửa sổ.

2.4.3. Các kiểu dữ liệu mở rộng từ các kiểu dữ liệu chuẩn trong C như:

LONG - Số nguyên dài

WORD - Số nguyên không dấu

DWORD - Số nguyên dài không dấu

BOOL - Kiểu logic gồm 2 giá trị là TRUE và FALSE

PSTR - Con trỏ trỏ tới một chuỗi ký tự (con trỏ ký tự)

LPSTR - Con trỏ xa trỏ tới một chuỗi ký tự (con trỏ ký tự xa)

FARPROC - Con trỏ xa trỏ tới một hàm (con trỏ hàm xa)

2.4.4. Các kiểu chuẩn của C mà chúng ta vẫn gặp trong C for DOS như: int, long, char, usigned, float, double, far , void.

Nhận xét: Chỉ các kiểu chuẩn của C viết bằng chữ thường, còn các kiểu khác đều viết bằng chữ hoa.

§3. CÁC TỆP NGUỒN CỦA MỘT CHƯƠNG TRÌNH

Một chương trình C for DOS chỉ chứa trên một tệp nguồn có đuôi C, còn một chương trình C for Windows chạy trên môi trường Visual C^{++} 6.0 thường được tổ chức trên 3 tệp nguồn.

3.1. Tệp chương trình đuôi C chứa các nội dung sau:

+ Các câu lệnh #include

+ Hàm WinMain , hàm cửa sổ (Window), các hàm hộp thoại (hàm DLG) và các hàm khác của chương trình.

3.2. Tệp tiêu đề đuôi H chứa các câu lệnh sau:

+ Các câu lệnh #define dùng để định nghĩa định danh (ID) của các nút lệnh của menu và của các hộp thoại.

Tệp tiêu đề được dùng trong các tệp chương trình (đuôi C) và tệp tài nguyên - resource (đuôi RC).

3.3. Tệp tài nguyên đuôi RC chứa các lệnh dùng để mô tả:

+ Menu

+ Các hộp thoại

+ Con trỏ

+ Biểu tượng

+ Các ảnh Bitmap

+ Các phím cấp tốc (phím nóng)

+ Các tài nguyên khác.

Nhận xét:

1. Cấu trúc của hàm WinMain, hàm cửa sổ và các hàm DLG sẽ trình bầy trong mục tiếp theo. Ở đây có thể nói trước đôi điều như sau:

+ Nội dung hàm WinMain gần như giống nhau đối với mọi chương trình, vì vậy chỉ cần soạn thảo hàm này một lần rồi sao cho các chương trình mới.

+ Các hàm cửa sổ và DLG có cấu trúc tương tự, dùng câu lệnh switch để phân loại và xử lý các thông điệp do Windows gửi tới. Vì vậy có thể soạn thảo một hàm mẫu (với khá đầy đủ các thông điệp) để dùng cho các chương trình khác nhau.

2. Tệp tài nguyên RC thường được viết đầu tiên để mô tả menu và các hộp thoại. Trong tệp này sẽ đặt định danh cho các nút lệnh của menu và hộp thoại. Định danh các nút lệnh cần được định nghĩa trong tệp tiêu đề (đuôi

H) và được sử dụng trong hàm cửa sổ và các hàm DLG để phân loại và xử lý các nút lệnh.

3. Tệp H là một tệp ngắn chỉ gồm các lệnh #define để định nghĩa định danh của các nút lệnh trong tệp RC.

Trong mục §4 sẽ minh hoạ cả 3 tệp nguồn thông qua một chương trình cụ thể.

§4. VÍ DỤ VỀ MỘT CHƯƠNG TRÌNH C FOR WINOWS

Trong mục này sẽ trình bầy các tệp nguồn của một chương trình C for Windows. Việc biên soạn, biên dịch và chạy chương trình trong môi trường Visual C^{++} 6.0 sẽ trình bầy trong mục tiếp theo.

4.1. Nội dung và thiết kế chương trình

4.1.1. Tên chương trình.

+ Ta đặt tên chương trình là VD1
+ Chương trình gồm 3 tệp nguồn là:

VD1.C Resource.rc Resource.h

4.1.2. Nội dung. Chương trình sẽ gồm các nội dung sau:

+ Tính diện tích tam giác theo đáy và chiều cao.
+ Tính diện tích hình tròn theo bán kính.
+ Vẽ một hình elip và in dòng chữ HELLO
+ Vẽ một hình chữ nhật
+ Xoá vùng làm việc của cửa sổ.
+ Kết thúc chương trình.

4.1.3. Thanh menu. Menu của chương trình như sau:

Vẽ và tính diện tích các hình		
Tính diện tích	Vẽ xoá	Kết thúc
Tam giác Hình tròn	Vẽ hình ellipse Vẽ chữ nhật Xoá CS	

Trong đó:

+ Vẽ và tính diện tích các hình - là tiêu đề cửa sổ chính
+ Tính diện tích - là menu con, gồm 2 nút lệnh:

- Tam giac
- Hình tròn

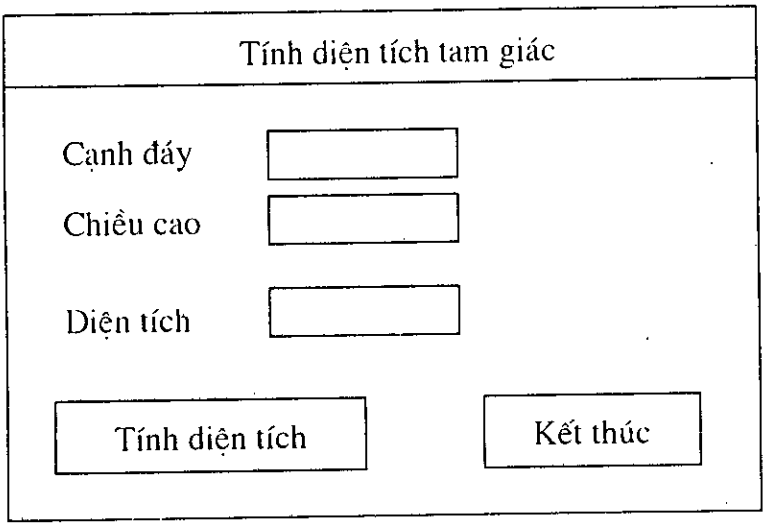
+ Vẽ xoá - là menu con, gồm 3 nút lệnh:
- Vẽ hình ellipse
- Vẽ chữ nhật
- Xoá CS

+ Kết thúc - là nút lệnh

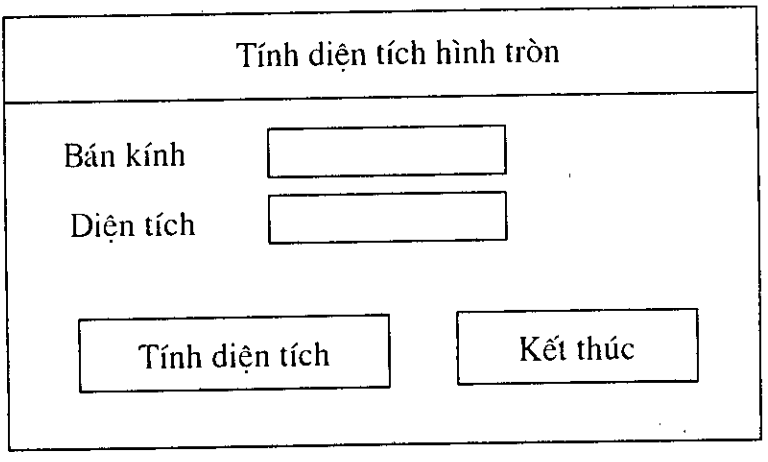
4.1.4. Các hộp thoại.

Có 2 hộp thoại:

+ Hộp thoại TGDLG dùng để tổ chức giao diện nhập dữ liệu và hiển thị kết quả cho chức năng tính diện tích tam giác, có dạng sau:



+ Hộp thoại HTDLG dùng để tổ chức giao diện nhập dữ liệu và hiển thị kết quả cho chức năng tính diện tích hình tròn, có dạng sau:



4.2. Nội dung tệp Resource.rc

```
// Tệp Resource.rc
#include <windows.h>
#include "resource.h"
// Menu
VD1 MENU DISCARDABLE
BEGIN
  POPUP "Tinh dien tich"
  BEGIN
    MENUITEM "Tam giac",          IDM_TG
    MENUITEM "Hinh tron",         IDM_HT
  END
  POPUP "Ve xoa"
  BEGIN
    MENUITEM "Ve hinh ellipse",   IDM_VE_ELIP
    MENUITEM "Ve chu nhat",       IDM_VE_CN
    MENUITEM "Xoa CS",            IDM_XOA
  END
  MENUITEM "Ket thuc",            IDM_KT
END

// Dialog
TGDLG DIALOG DISCARDABLE  0, 0, 186, 111
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Tinh dien tich tam giac"
FONT 8, "MS Sans Serif"
BEGIN
 LTEXT        "Canh day",IDC_STATIC_DAY,13,11,31,8
 EDITTEXT      IDC_EDIT_DAY,71,9,40,14,ES_AUTOHSCROLL
 LTEXT        "Chieu cao",IDC_STATIC_CAO,15,35,33,8
 EDITTEXT      IDC_EDIT_CAO,67,32,40,14,ES_AUTOHSCROLL
 LTEXT        "Dien tich",IDC_STATIC_DTTG,15,59,29,8
 EDITTEXT      IDC_EDIT_DTTG,63,56,40,14,ES_AUTOHSCROLL
 PUSHBUTTON   "Tinh dien tich",IDC_BUTTON_TINH_DTTG,15,80,50,14
  PUSHBUTTON     "Ket thuc",IDC_BUTTON_KT_TG,101,80,50,14
END
```

```
HTDlg DIALOG DISCARDABLE  0, 0, 186, 95
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Tinh dien tich hinh tron"
FONT 8, "MS Sans Serif"
BEGIN
 LTEXT        "Ban kinh",IDC_STATIC_BK,25,14,29,8
 EDITTEXT      IDC_EDIT_BK,79,11,40,14,ES_AUTOHSCROLL
 LTEXT        "Dien tich",IDC_STATIC_DTHT,23,37,29,8
 EDITTEXT      IDC_EDIT_DTHT,71,32,40,14,ES_AUTOHSCROLL
 PUSHBUTTON  "Tinh dien tich",IDC_BUTTON_TINH_DTHT, 19,64,50,14
   PUSHBUTTON     "Ket thuc",IDC_BUTTON_KT_HT,107,67,50,14
END
```

4.3. Nội dung tệp Resource.h

```
   // Tệp Resource.h
#define IDC_EDIT_DAY                1000
#define IDC_STATIC_CAO              1001
#define IDC_EDIT_CAO                1002
#define IDC_STATIC_DTTG             1003
#define IDC_EDIT_DTTG               1004
#define IDC_BUTTON_TINH_DTTG        1005
#define IDC_BUTTON_KT_TG            1006
#define IDC_STATIC_BK               1007
#define IDC_EDIT_BK                 1008
#define IDC_STATIC_DTHT             1009
#define IDC_EDIT_DTHT               1010
#define IDC_BUTTON_TINH_DTHT        1011
#define IDC_BUTTON_KT_HT            1012
#define IDM_TG                     40001
#define IDM_HT                     40002
#define IDM_VE_ELIP                40003
#define IDM_VE_CN                  40004
#define IDM_XOA                    40005
#define IDM_KT                     40006
#define IDC_STATIC_DAY              -1
```

4.4. Nội dung tệp VD1.C

```
   // Tệp VD1.C
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "resource.h"
char ten_ct[] = "vd1";
HINSTANCE hInst;
HWND hWndMain;
BOOL ve_cn = FALSE;
LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
BOOL CALLBACK TGDlgProc(HWND,UINT,WPARAM,LPARAM);
BOOL CALLBACK HTDlgProc(HWND,UINT,WPARAM,LPARAM);
//Hàm WinMain
int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,
                   LPSTR lpszCmdLine,int nCmdShow)
 {
  MSG msg;  WNDCLASS wc;
  hInst = hInstance;
  memset(&wc, 0, sizeof(WNDCLASS));
     wc.style=CS_HREDRAW|CS_VREDRAW;
     wc.lpfnWndProc = WndProc;
     wc.hInstance = hInst;
     wc.lpszClassName = ten_ct;
     wc.lpszMenuName = ten_ct;
     wc.cbClsExtra = 0;
     wc.cbWndExtra = 0;
     wc.hIcon = LoadIcon(NULL,IDI_APPLICATION);
     wc.hCursor = LoadCursor(NULL,IDC_ARROW);
     wc.hbrBackground= GetStockObject(WHITE_BRUSH);
     if(!RegisterClass(&wc))
      {
        MessageBeep(0xFFFFFFFF);
        MessageBox(NULL,"Loi DK lop",NULL,MB_OK);
        return 0;
      }
```

```
    hWndMain= CreateWindow(ten_ct, "Ve va tinh dien tich cac hinh",
                        WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, HWND_DESKTOP,
                        NULL, hInst,NULL);
   if(hWndMain==NULL)
     {
       MessageBox(NULL,"Loi tao CS",NULL,MB_OK);
       return 0;
     }
   ShowWindow(hWndMain,nCmdShow);
         UpdateWindow(hWndMain);
   while(GetMessage(&msg,NULL,0,0))
     {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
     }
    UnregisterClass(ten_ct,hInst);
   return msg.wParam;
  }
//Hàm cửa sổ
 LRESULT CALLBACK WndProc(HWND hWnd,UINT Message,
                                  WPARAM wParam, LPARAM lParam)
   {
    HDC hDC;
         PAINTSTRUCT ps;
         RECT rect;
    switch(Message)
     {
      case WM_COMMAND:
       switch(wParam)
        {
         case IDM_TG:
            DialogBox(hInst,"TGDlg",hWnd,TGDlgProc);
           break;
         case IDM_HT:
            DialogBox(hInst,"HTDlg",hWnd,HTDlgProc);
          break;
```

```
        case IDM_VE_ELIP:
          hDC=GetDC(hWnd);
          GetClientRect(hWnd,&rect);
          Ellipse(hDC,20,20,rect.right-20,rect.bottom-20);
          TextOut(hDC,rect.right/2,rect.bottom/2,"HELLO",5);
          ReleaseDC(hWnd,hDC);
          break;
        case IDM_VE_CN:
          ve_cn = TRUE;
        InvalidateRect(hWnd,NULL,TRUE);
          break;
        case IDM_XOA:
          ve_cn= FALSE;
          InvalidateRect(hWnd,NULL,TRUE);
          break;
        case IDM_KT:
          PostMessage(hWnd,WM_CLOSE,0,0L);
          break;
        default:
          return DefWindowProc(hWnd,Message,wParam,lParam);
          break;
      }
    break;
  case WM_PAINT:
    memset(&ps,0x00,sizeof(PAINTSTRUCT));
    hDC = BeginPaint(hWnd,&ps);
    if(ve_cn)
    {
      GetClientRect(hWnd,&rect);
      Rectangle(hDC,20,20,rect.right-20,rect.bottom-20);
    }
     EndPaint(hWnd,&ps);
    break ;
  case WM_CLOSE:
    DestroyWindow(hWnd);
    PostQuitMessage(0);
    break ;
```

```
        default:
            return DefWindowProc(hWnd,Message,wParam,lParam);
    }
   return 0L;
  }
//Hàm quản lý hộp thoại TGDlg
 BOOL CALLBACK TGDlgProc(HWND hDlg,UINT Message,WPARAM
                                    wParam,LPARAM lParam)
  {
   switch(Message)
    {
    case WM_COMMAND:
      switch(wParam)
       {
        case IDC_BUTTON_TINH_DTTG:
          {
           double a,h,dt; char tg[20];
           GetDlgItemText(hDlg,IDC_EDIT_DAY,tg,19);
           a=atof(tg);
           GetDlgItemText(hDlg,IDC_EDIT_CAO,tg,19);
           h=atof(tg);
           dt=a*h/2;
           sprintf(tg,"%0.2f",dt);
           SetDlgItemText(hDlg,IDC_EDIT_DTTG,tg);
          }
         break;
        case IDC_BUTTON_KT_TG:
          EndDialog(hDlg,FALSE);
          break;
        default:
          return FALSE;
       }
      break;
    case WM_CLOSE:
      EndDialog(hDlg,FALSE);
      break;
    case WM_INITDIALOG:
```

```
        SetFocus(hDlg);
        break;
      default:
        return FALSE;
    }
  return TRUE;
 }
//Hàm quản lý hộp thoại HTDlg
 BOOL CALLBACK HTDlgProc(HWND hDlg,UINT Message,WPARAM
                                    wParam,LPARAM lParam)
 {
  switch(Message)
   {
    case WM_COMMAND:
      switch(wParam)
       {
        case IDC_BUTTON_TINH_DTHT:
         {
           double r,dt; char tg[20];
           GetDlgItemText(hDlg,IDC_EDIT_BK,tg,19);
           r=atof(tg);
           dt=3.14*r*r;
           sprintf(tg,"%0.2f",dt);
           SetDlgItemText(hDlg,IDC_EDIT_DTHT,tg);
         }
         break;
        case IDC_BUTTON_KT_HT:
           EndDialog(hDlg,FALSE);
           break;
        default:
           return FALSE;
       }
      break;
    case WM_CLOSE:
      EndDialog(hDlg,FALSE);
      break;
    case WM_INITDIALOG:
      SetFocus(hDlg);
```

```
        break;
      default:
        return FALSE;
    }
  return TRUE;
}
```

§5. BIÊN SOẠN, BIÊN DỊCH VÀ CHẠY CHƯƠNG TRÌNH TRONG MÔI TRƯỜNG VISUAL C++ 6.0

Tạo thư mục D:\WVC6\C-Win để chứa các ứng dụng (chương trình). Để vào môi trường của Visual C++ 6.0, ta có thể sử dụng một trong hai cách sau:

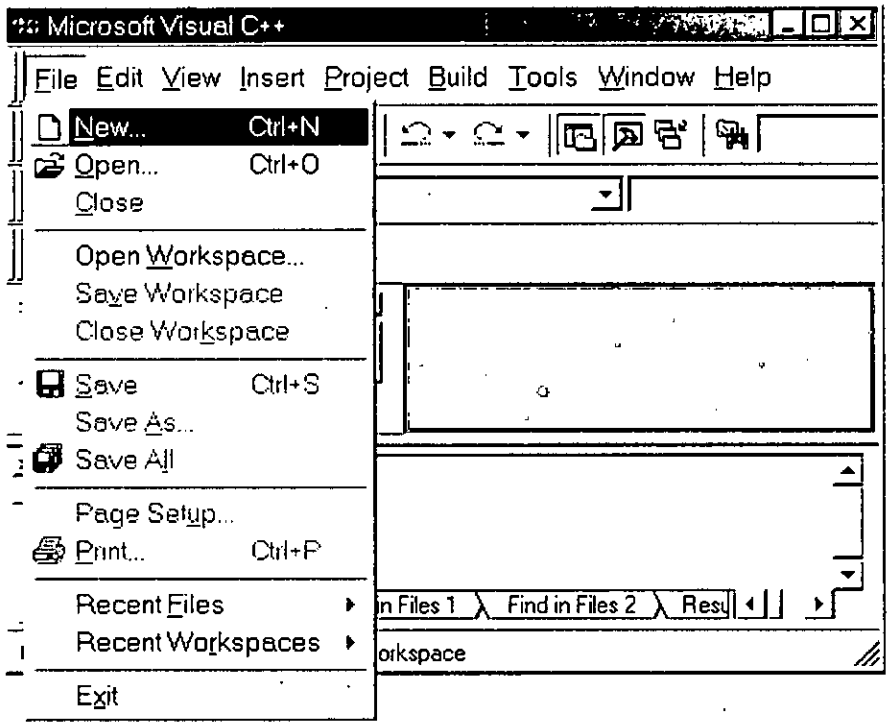
1. Chọn Start, programs, Microsoft Visual Studio 6.0, Microsoft Visual C++ 6.0.

2. Double click vào biểu tượng Microsoft Visual C++ 6.0 trên desktop.

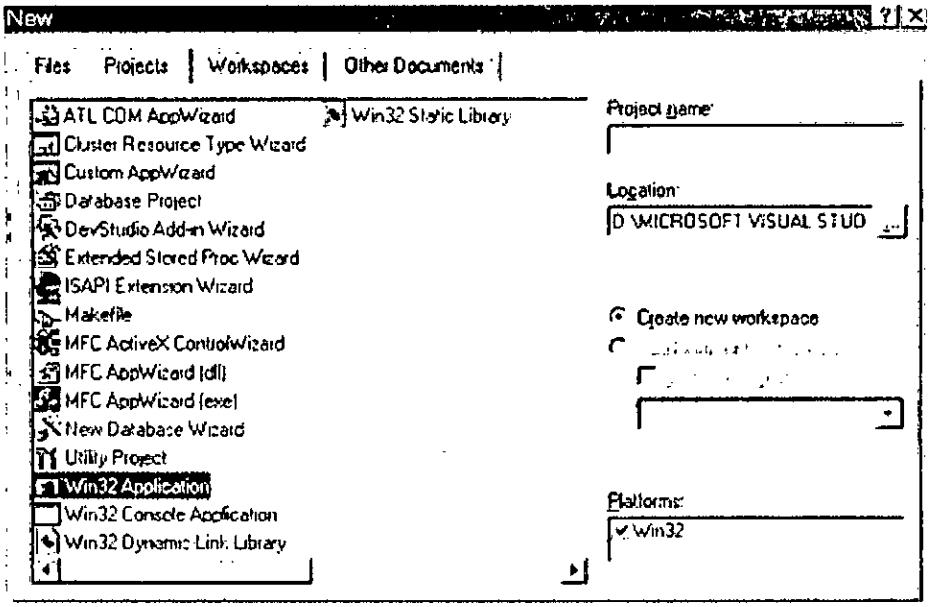
Để biên soạn, biên dịch và chạy chương trình nêu trong mục §4, hãy làm theo các bước sau:

Bước 1. Tạo một Dự án (Project) mới

+ Chọn File, New



Chọn trang Project, Win32 Application:

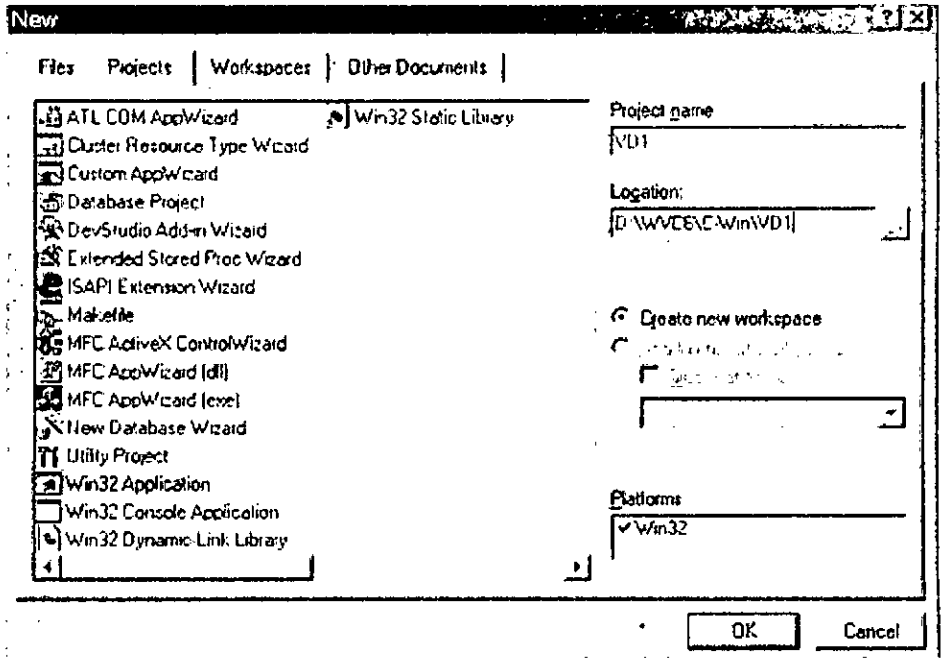


+ Soạn thảo nội dung như sau vào 2 mục:

Project name: VD1 - Tên project là VD1.dsw

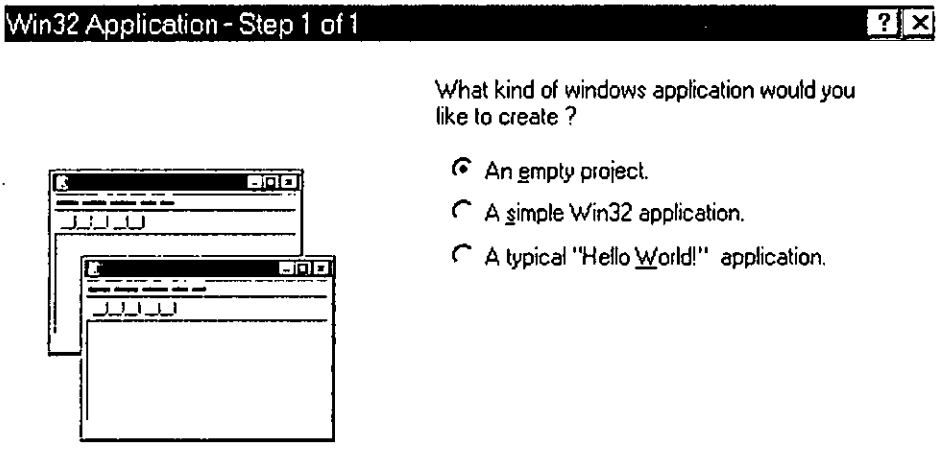
Location: D:\WVC6\C-Win\VD1 - Thư mục chứa project

(Thư mục sẽ tự động được tạo tạo, nếu chưa có)



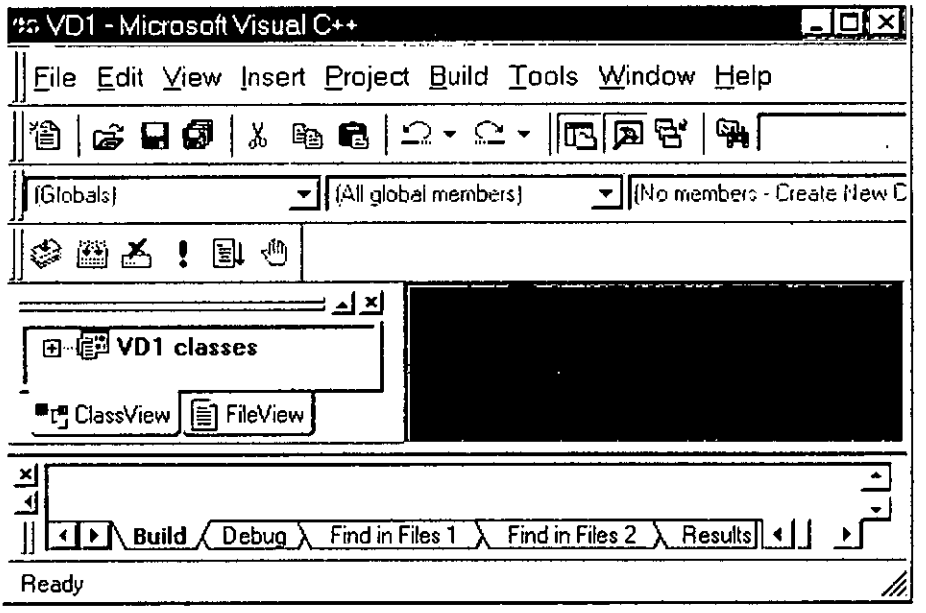
+ Bấm OK

+ Chọn mục An empty project (mặc định)



+ Bấm Finish và OK

Kết quả: Tạo tệp dự án (project) VD1.dsw trong thư mục D:\WVC6\C-Win\VD1 và nhận được cửa sổ sau:

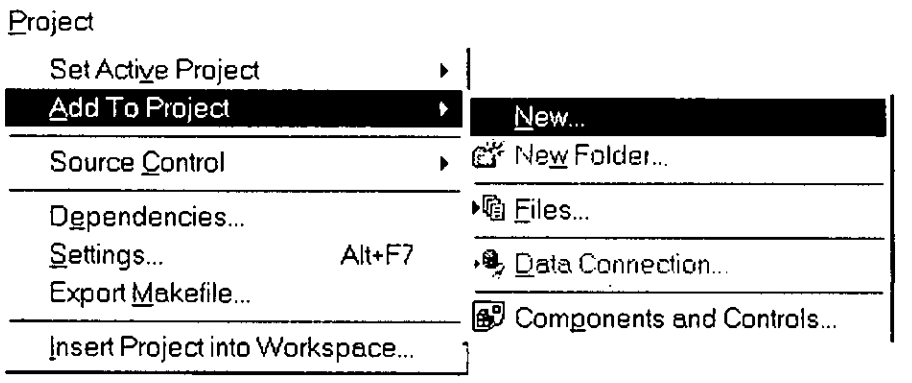


Tệp dự án sẽ quản lý các tệp khác của một ứng dụng viết bằng C for Windows.

Bước 2. Bổ sung vào dự án 3 tệp nguồn

2.1. Bổ sung tệp Resource.rc

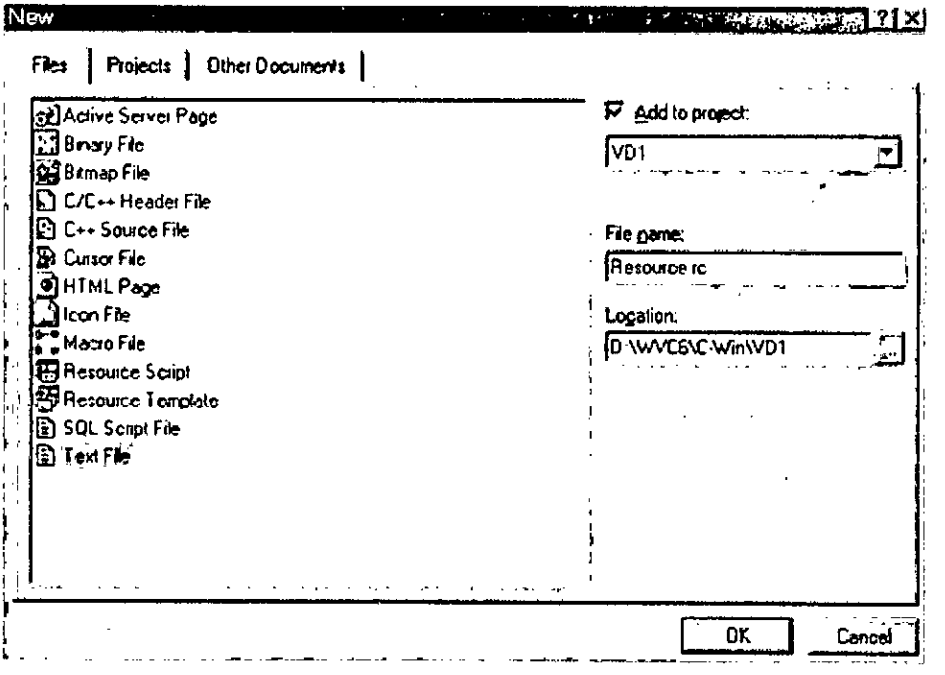
+ Chọn Project, Add To Project, New :



+ Chọn mục Text File

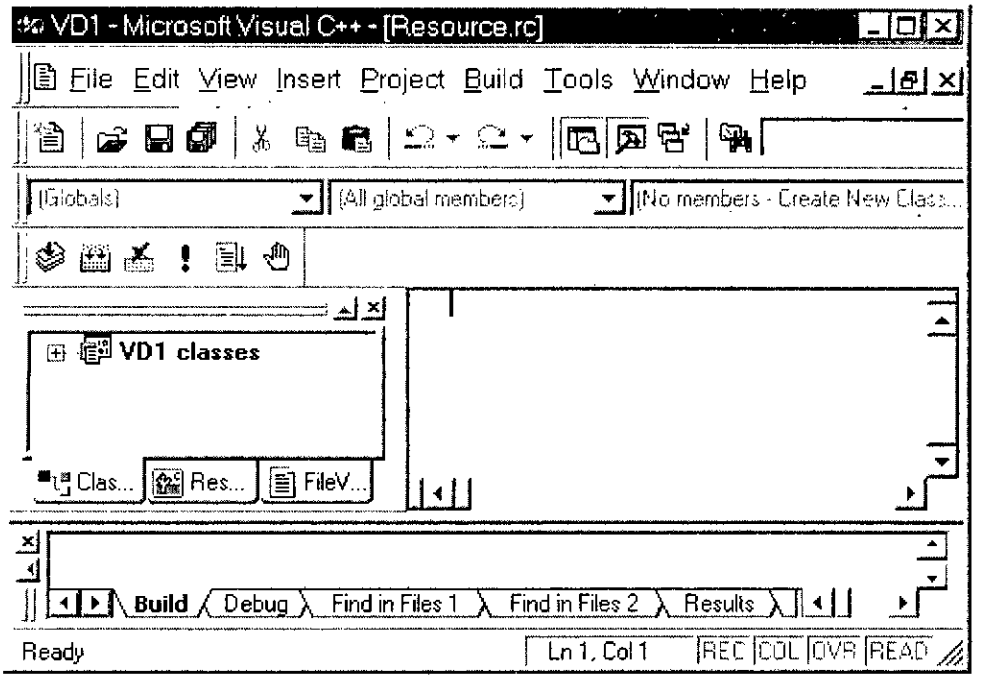
+ Đặt tên tệp :

File name: Resource.rc

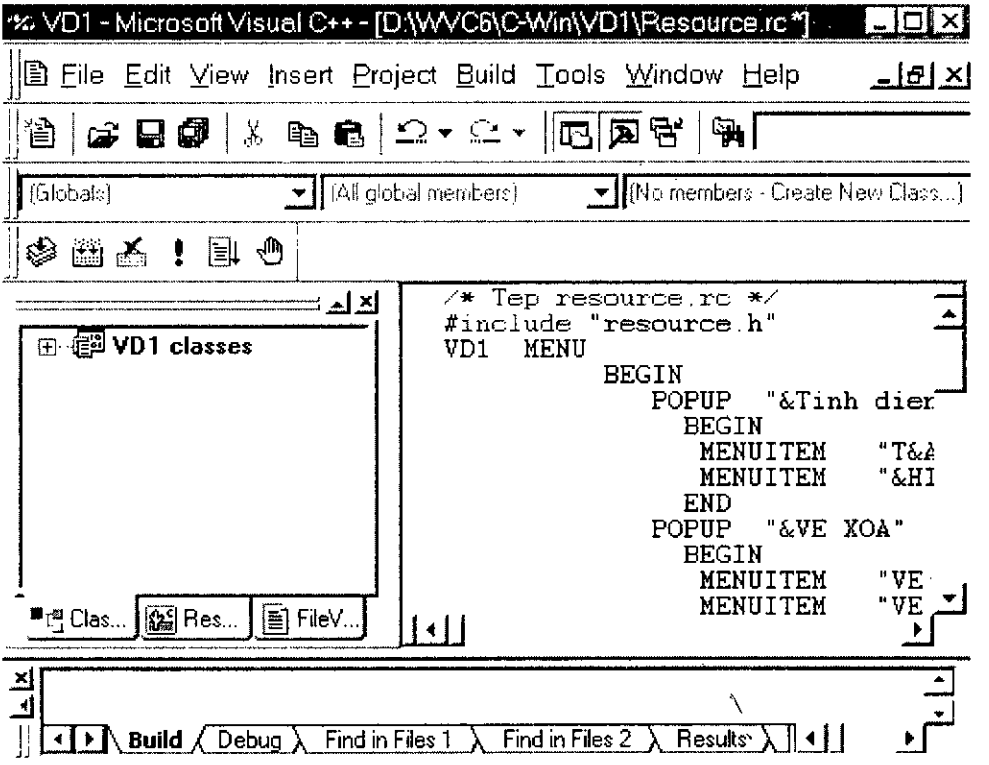


+ Bấm OK

Kết quả: Nhận được cửa sổ soạn thảo tệp Resource.rc



+ Soạn thảo tệp Resource.rc theo nội dung nêu trong mục §4:



2.2. Bổ sung tệp Resource.h

Tiến hành tương tự, chỉ khác ở chỗ đặt tên tệp là Resource.h

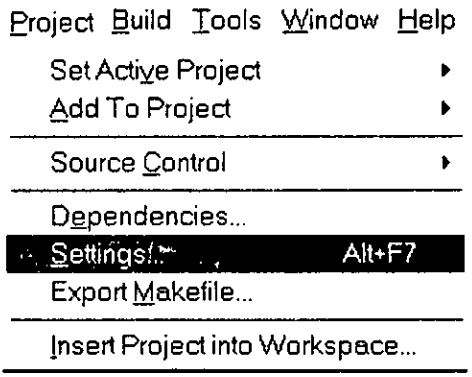
2.3. Bổ sung tệp VD1.C

Tiến hành tương tự, chỉ khác ở chỗ đặt tên tệp là VD1.C

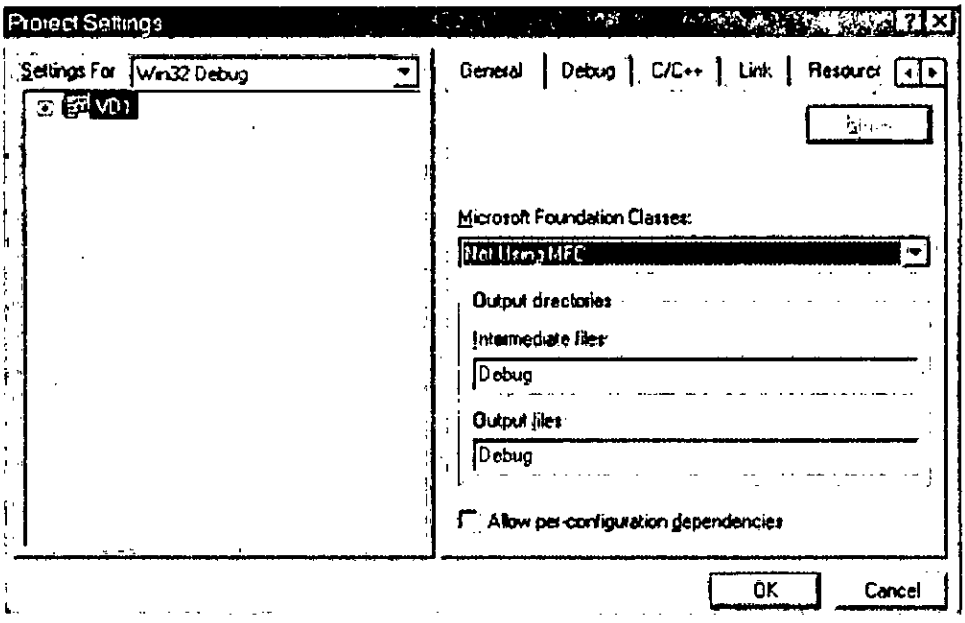
Bước 3. Dịch và chạy chương trình

+ Chọn :

Project, Settings



+ Chọn trang General và chọn mục Microsoft Foundation Classes như sau: Microsoft Foundation Classes: Not Using MFC

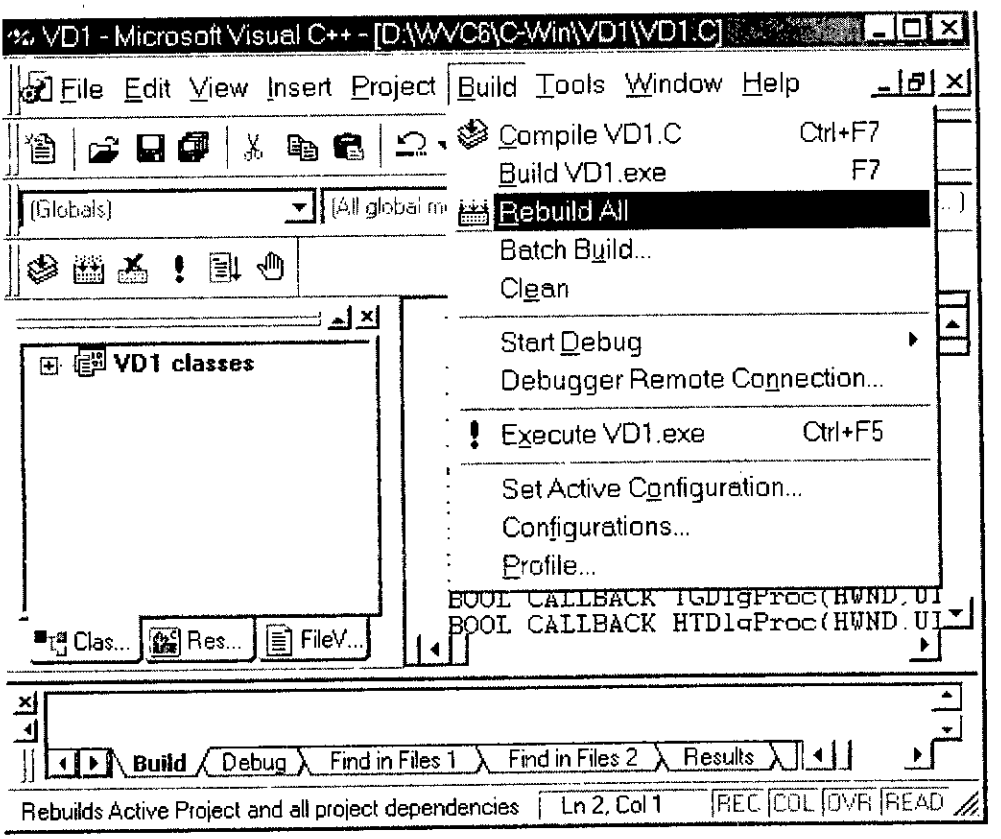


Chọn OK

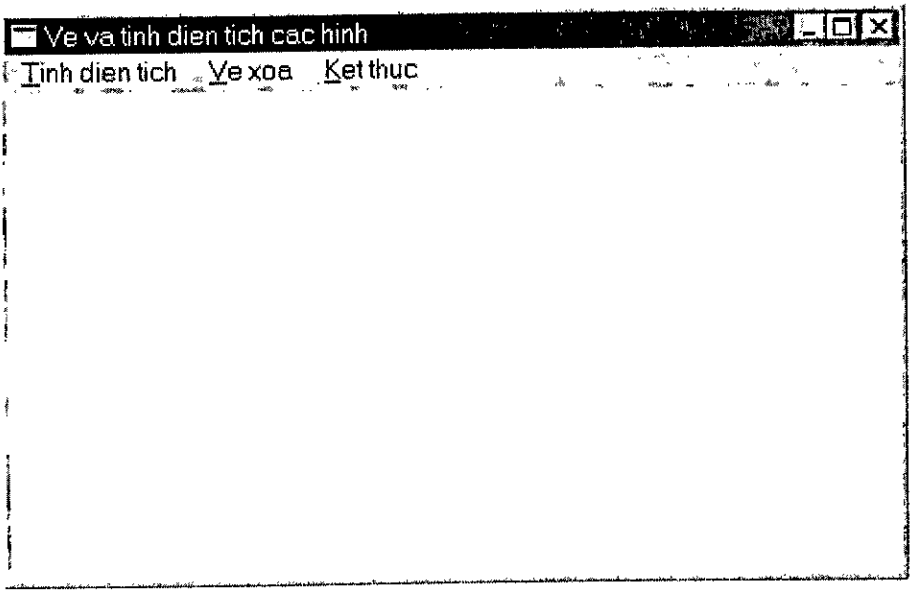
+ Chọn menu Build, sau đó lần lượt chọn các mục:

Compile VD1.C (Dịch tệp VD1.C)

Rebuild All (Dịch, liên kết và tạo ra tệp VD1.EXE)



Execute VD1.EXE (Thực hiện tệp chương trình VD1.EXE), kết quả:



§6. SỬ DỤNG CÔNG CỤ CỦA VISUAL C++ ĐỂ TẠO MENU VÀ HỘP HỘI THOẠI

Như đã nói ở §3, một chương trình C for Windows gồm 3 tệp. Các tệp rc và h dùng để mô tả menu và hộp thoại (cũng như các tài nguyên khác). Các tệp này có thể được tự động tạo ra bằng cách sử dụng các công cụ của Visual C++ 6.0. Khi xây dựng chương trình, chỉ cần thiết kế menu và các hội thoại, sau đó dùng công cụ để tạo chúng. Chỉ còn một tệp duy nhất cần soạn thảo là tệp chương trình đuôi C.

Khi thiết kế menu cần chỉ rõ:

+ Tên menu
+ Tiêu đề các menu con
+ Tiêu đề và định danh của từng nút lệnh

Khi thiết kế một hộp hội thoại cần chỉ rõ:

+ Tên hộp thoại
+ Tiêu đề các nhãn (Static Text)
+ Định danh của từng nút hộp soạn thảo (Edit Box)
+ Tiêu đề và định danh của từng nút lệnh (Push Button)

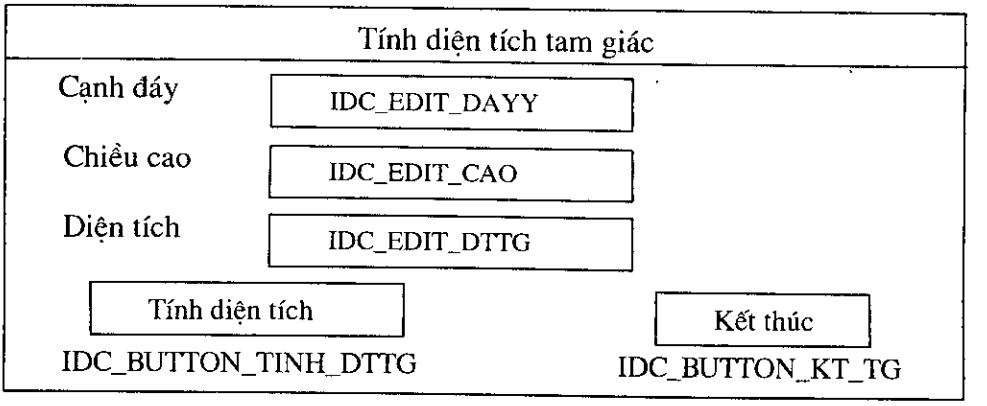
Ví dụ menu và các hộp thoại trong mục §4, có thể thiết kế như sau:

Name: "VD1"

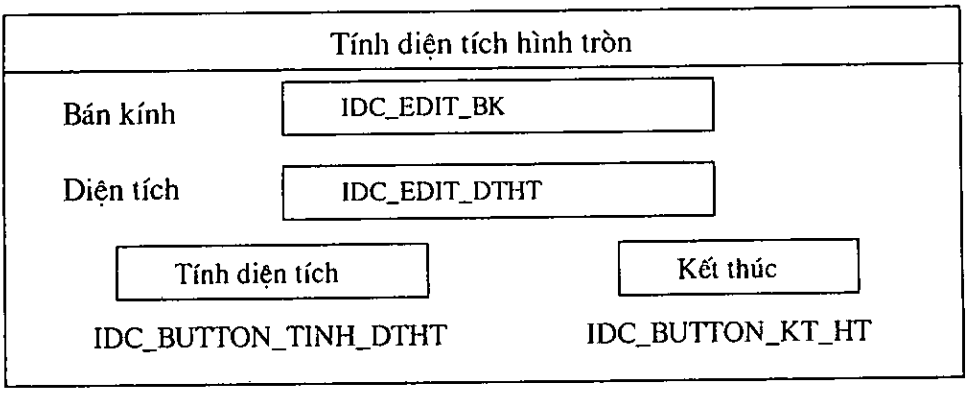
Vẽ và tính diện tích các hình		
Tính diện tích	Vẽ xoá	Kết thúc (IDM_KT)
Tam giác (IDM_TG) Hình tròn (IDM_HT)	Vẽ hình ellipse (IDM_VE_ELIP) Vẽ chữ nhật (IDM_VE_CN) Xoá CS (IDM_XOA)	

Chú ý: Cần phân biệt định danh và tiêu đề. Định danh sẽ được định nghĩa trong tệp h và được sử dụng để viết các câu lệnh. Còn tiêu đề chỉ có tác dụng hiển thị trên màn hình. Chẳng hạn trong ví dụ trên thì: IDM_TG là định danh của nút lệnh, còn tiêu đề là: Tam giác

Name: "TGDlg"



Name: "HTDlg"

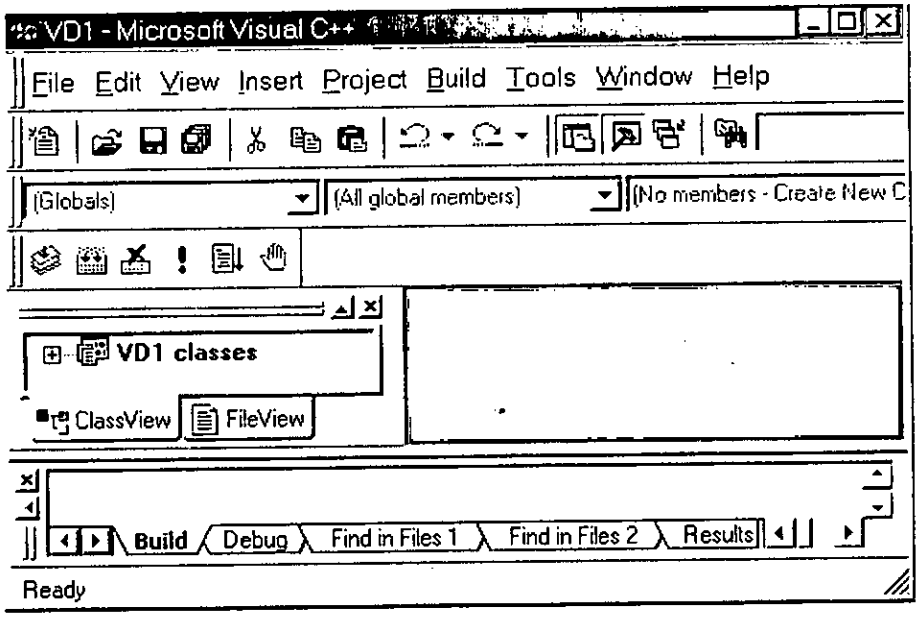


Sau khi đã thiết kế giao diện (menu và các hộp thoại), có thể xây dựng ứng dụng theo các bước sau:

Bước 1. Tạo một Dự án (Project) mới (xem Bước 1 trong §5)

+ Chọn File, New, Project, Win32 Application
  Project name: VD1
  Location: D:\WVC6\C-Win\VD1
  OK
+ Chọn
  An empty project
  Finish
  OK

Kết quả ta có cửa sổ chương trình:



Bước 2. Sử dụng công cụ để tạo menu, hộp thoại và đưa chúng vào Dự án

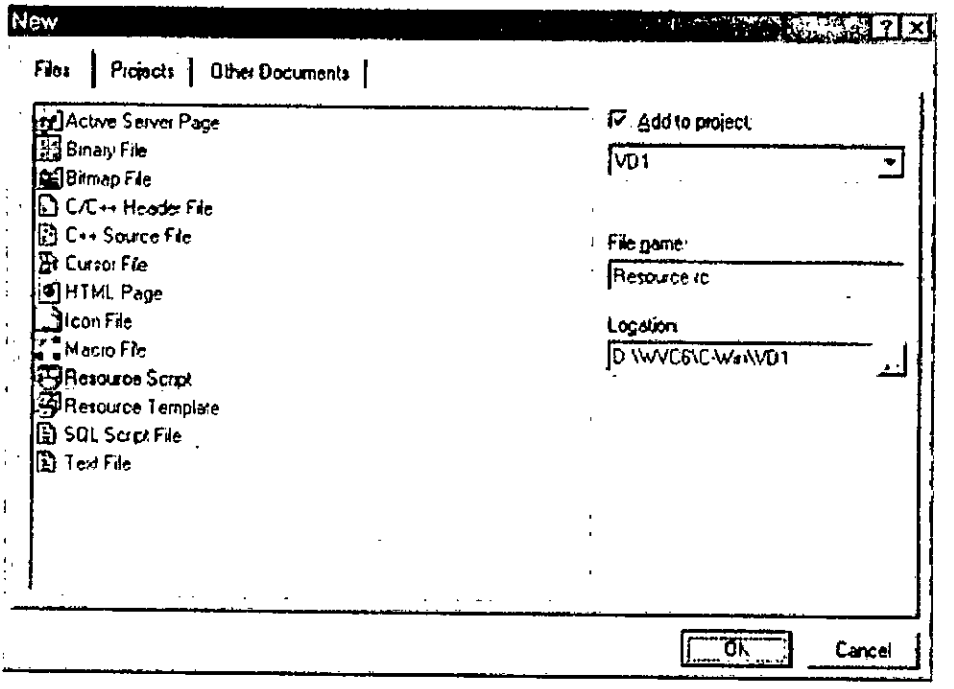
2.1. Bổ sung tệp Resource.rc

+ Chọn Project, Add To Project, New, Resource Script

+ Soạn thảo

Add to project: VD1 (có sẵn)

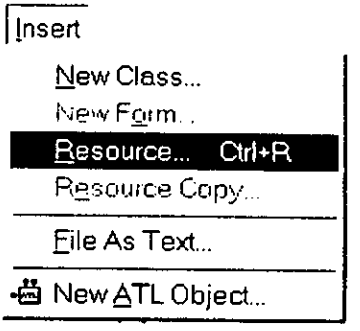
File name: Resource (hoặc Resource.rc)



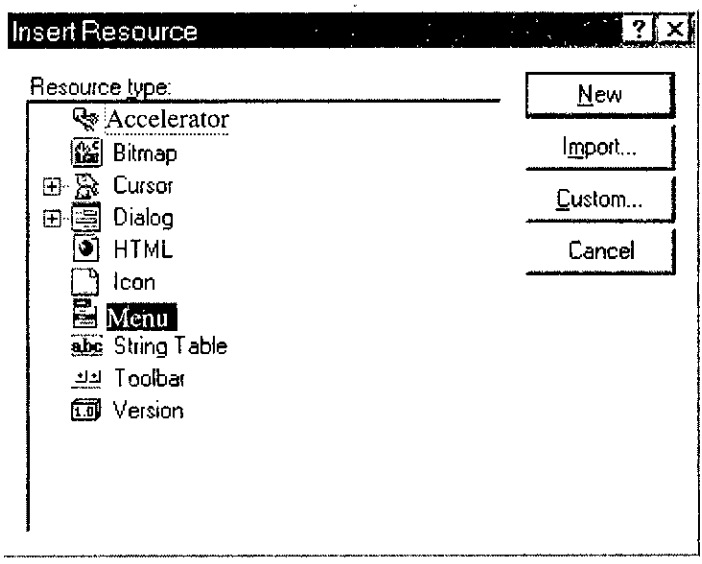
+ OK

2.2. Bổ sung một menu (theo thiết kế bên trên) vào tệp Resource.rc

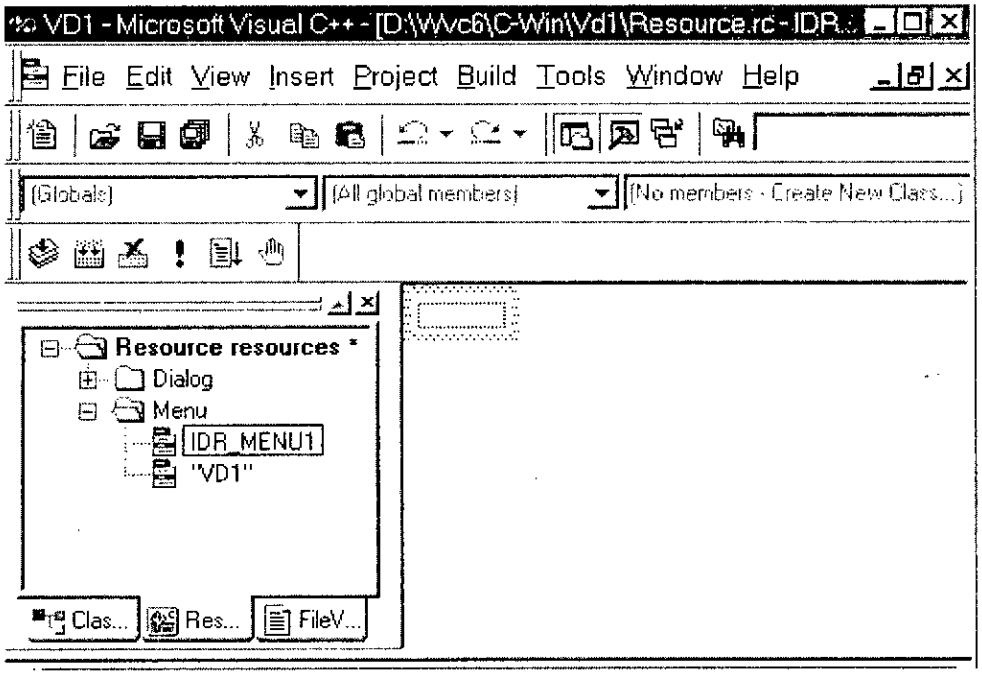
+ Chọn Resource.rc, Insert, Resource (hoặc bấm phải chuột rồi chọn mục Insert)



Kết quả nhận được cửa sổ Insert Resource như sau:

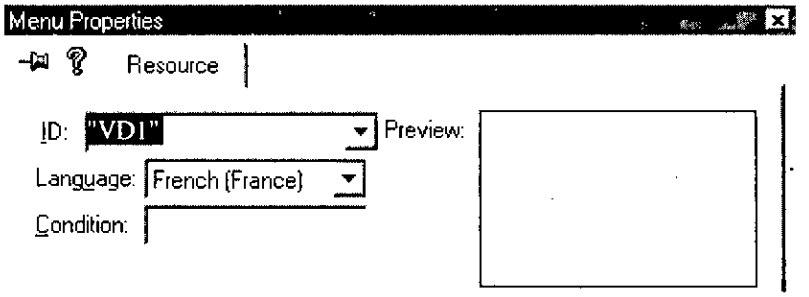


+ Trong cửa sổ Insert Resource, chọn mục Menu, rồi bấm nút New
Kết quả nhận được một menu rỗng:



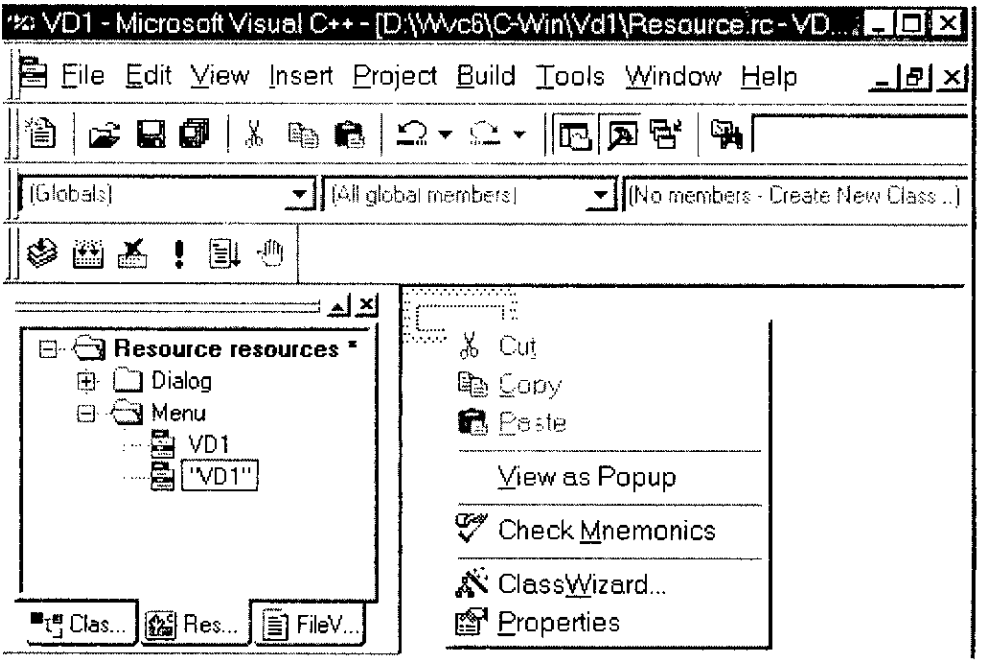
+ Đặt tên menu bằng cách bấm phải chuột, chọn mục Properties
Kết quả nhận được cửa sổ Menu Properties
Trong hộp ID của cửa sổ này, soạn thảo tên menu: "VD1"

Kết quả nhận được cửa sổ:

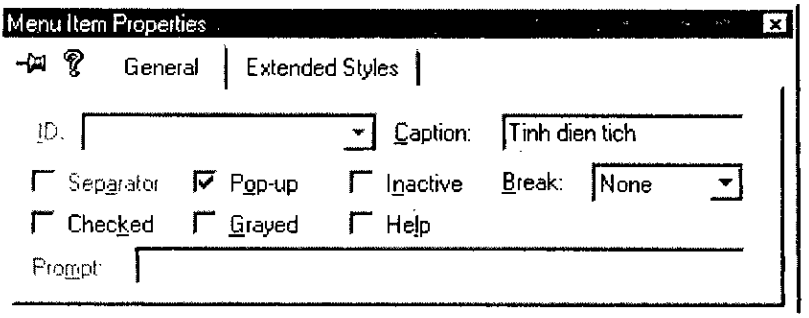


Chú ý: Tên menu phải đặt trong 2 dấu nháy kép (xem ví dụ trên)

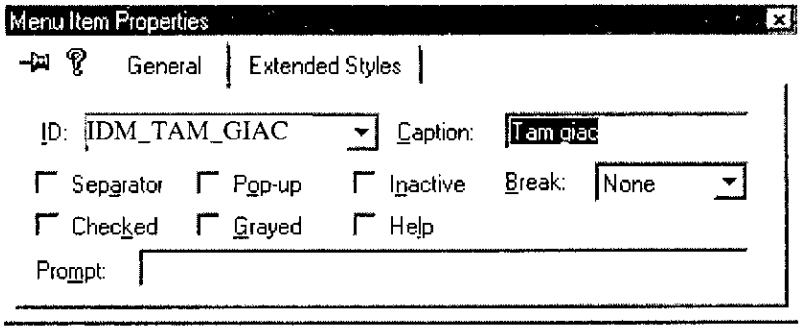
+ Thiết kế các menu con bằng cách mở cửa sổ Menu Item Properties (chọn một mục menu, bấm phải chuột, rồi chọn mục Properties).



Trong cửa sổ này, chọn ô Pop-up và soạn thảo tiêu đề của menu con trong hộp Caption (ví dụ: Tinh dien tich)



+ Thiết kế các nút lệnh bằng cách mở cửa sổ Menu Item Properties (chọn một mục menu, bấm phải chuột, rồi chọn mục Properties). Trong cửa sổ này, không chọn ô Pop-up, soạn thảo định danh của nút lệnh trong hộp ID và soạn thảo tiêu đề của nút lệnh trong hộp Caption.

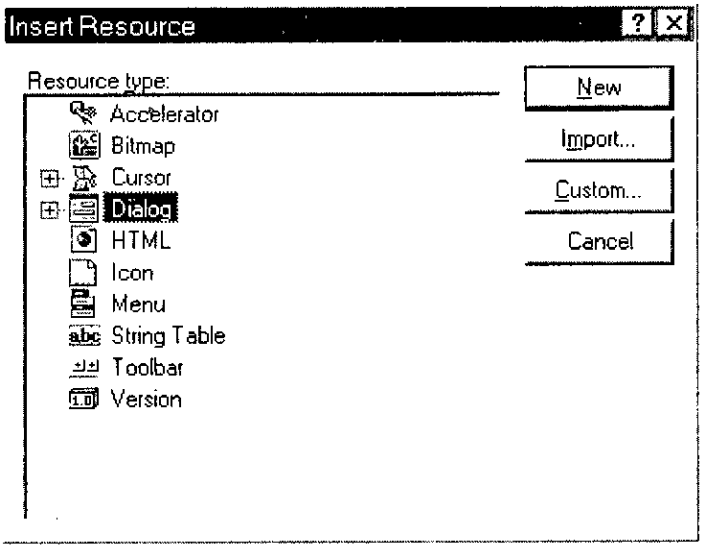


2.3. Bổ sung hộp thoại TGDlg (theo thiết kế bên trên) vào tệp Resource.rc

+ Chọn Resource.rc, Insert, Resource (hoặc bấm phải chuột rồi chọn mục Insert)

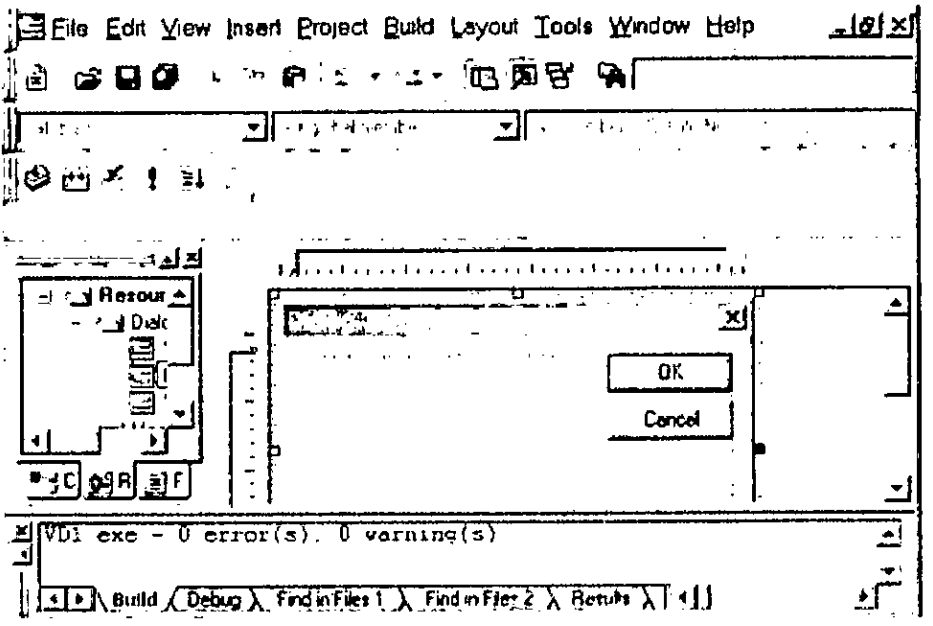
Kết quả nhận được cửa sổ: Insert Resource

+ Trong cửa sổ Insert Resource, chọn mục Dialog



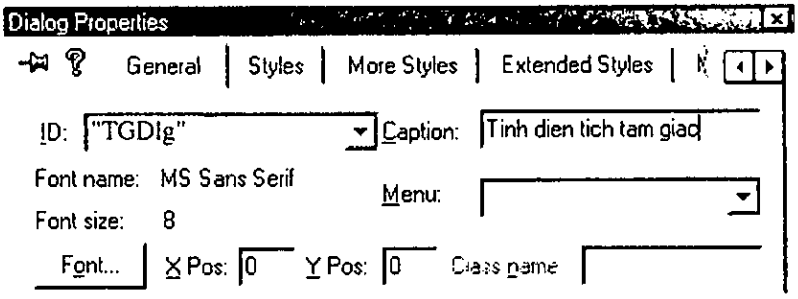
rồi bấm nút New

Kết quả nhận được một hộp thoại với tiêu đề: Dialog, có sẵn 2 nút lệnh OK và Cancel

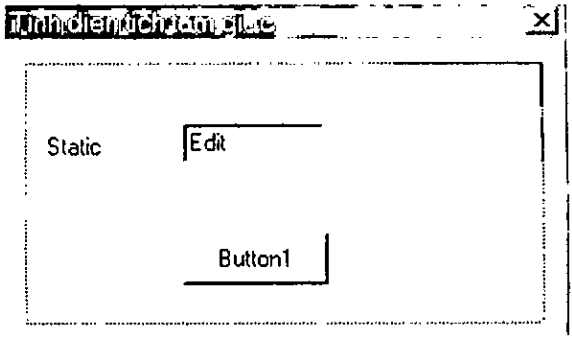


+ Xoá các nút lệnh OK và Cancel

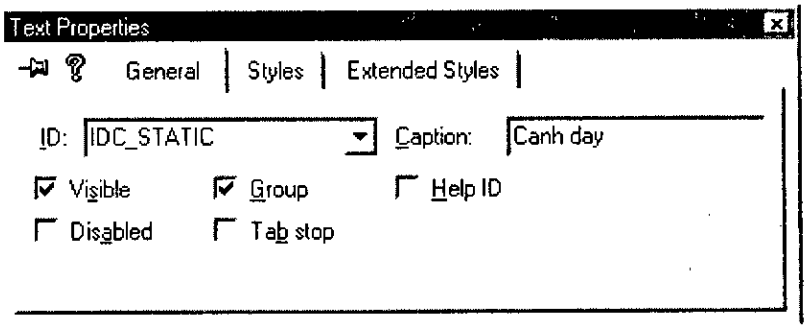
+ Đặt lại tiêu đề (cho hộp thoại) bằng cách bấm phải chuột, chọn mục Properties. Kết quả nhận được cửa sổ Dialog Properties. Trong cửa sổ này, tại hộp ID soạn thảo tên "TGDlg" và tại hộp Caption soạn thảo tiêu đề: Tính diện tích tam giác.



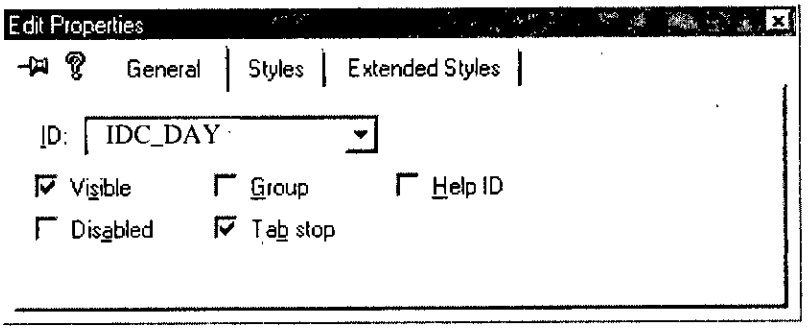
+ Dùng hộp công cụ để tạo các nhãn (Static Text), các hộp soạn thảo (Edit Box) và các nút lệnh (Push Button).



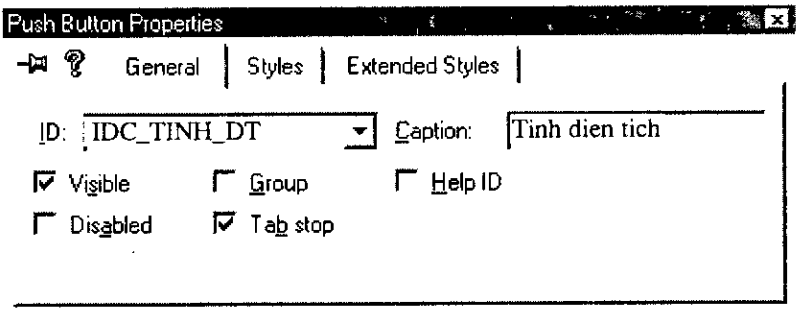
Đối với nhãn, đầu tiên mở cửa sổ Text Properties. Trong cửa sổ này hãy bỏ qua hộp ID và sử dụng hộp Caption để soạn thảo tiêu đề cho nhãn.



Đối với Edit Box, đầu tiên mở cửa sổ Edit Properties. Trong cửa sổ này dùng hộp ID để soạn thảo định danh cho Edit Box.



Đối với nút lệnh, đầu tiên mở cửa sổ Push Button Properties. Trong cửa sổ này dùng hộp ID để soạn thảo định danh cho nút lệnh và dùng hộp Caption để soạn thảo tiêu đề cho nút lệnh, ví dụ:



2.4. Bổ sung hộp thoại HTDlg(theo thiết kế bên trên) vào tệp Resource.rc

+ Tiến hành tương tự như trong 2.3 bên trên

2.5. Ghi nội dung đã thiết kế menu và các hộp thoại (chọn File, Save All hoặc dùng biểu tượng Save All). Kết quả sẽ nhận được tệp Resource.h

Bước 3. Bổ sung tệp VD1.C vào Dự án (xem Bước 2 trong §5)

+ Chọn Project, Add To Project, New, Text File
+ Soạn thảo

  Add to project: VD1 (có sẵn)

  File name: VD1.C
+ OK
+ Soạn thảo tệp VD1.C

Bước 4. Dịch và chạy chương trình (xem Bước 3 trong §5)

+ Chọn

  Project, Settings, General

  Microsoft Foundation Classes: Not Using MFC
+ Chọn menu Build, sau đó lần lượt chọn các mục:

  Compile VD1.C (Dịch tệp VD1.C)

  Rebuild All (Dịch, liên kết và tạo ra tệp VD1.EXE)

  Execute VD1.EXE (Thực hiện tệp chương trình VD1.EXE)

CHƯƠNG 2

HÀM WINMAIN VÀ CỬA SỔ

§1. CẤU TRÚC CỦA CỬA SỔ

+ Cửa sổ chính dùng để làm khung giao diện cho một chương trình (ứng dụng). Hầu hết mỗi chương trình có một cửa sổ chính, vì vậy khi thiết kế chương trình trước hết phải nghĩ đến xây dựng cửa sổ.

+ Để quản lý và làm việc với cửa sổ, các chương trình Windows sử dụng một biến kiểu HWND để lưu trữ chỉ danh (Handle) cửa sổ. Hàm CreateWindow (xem mục mục §3 dưới đây) sẽ xây dựng và cho chỉ danh của cửa sổ. Hàm thường được dùng theo mẫu sau:

```
HWND hWnd;
hWnd = CreateWindow( ... );
```

+ Đọc giả chắc đã quen biết và từng sử dụng các thành phần của cửa sổ. Ở đây, trên phương diện lập trình sẽ giới thiệu 2 bộ phận quan trọng là: thanh menu (menu bar) và vùng làm việc (client area).

+ Thông thường cửa sổ chính có một thanh menu (nói cho gọn là menu) để thể hiện các chức năng của chương trình. Có thể xây dựng nhiều menu, sau đó tuỳ theo yêu cầu bài toán mà thay đổi menu gắn cho cửa sổ chương trình. Cũng có thể xây dựng chương trình mà cửa sổ không có menu.

+ Vùng làm việc của cửa sổ là một hình chữ nhật dùng để hiển thị các chuỗi ký tự và các hình vẽ đồ hoạ. Vị trí và kích thước của vùng làm việc có thể thay đổi bởi người dùng khi chương trình đang hoạt động. Để xác định vị trí và kích thước vùng làm việc của cửa có chỉ danh hWnd ta dùng các câu lệnh sau:

```
RECT r ; // Khai báo biến cấu trúc r kiểu RECT
GetClientRect (hWnd, &r) ;
```

Kết quả: Giá trị các trường r.left, r.right, r.bottom và r.top sẽ cho biết vị trí vùng làm việc trên màn hình. Cụ thể:

(r.left, r.top) là toạ độ của điểm trên - trái
(r.right, r.bottom) là toạ độ của điểm dưới - phải
Đơn vị đo là Pixel

§2. ĐĂNG KÝ MỘT LỚP CỬA SỔ

+ Trước khi xây dựng một cửa sổ bằng hàm CreateWindow cần thực hiện việc đăng ký một lớp cửa sổ.

+ Để đăng ký một lớp cửa sổ thường làm theo các bước sau:

- Khai báo một biến cấu trúc kiểu WNDCLASS
- Gán các giá trị đặc trưng cho lớp cửa sổ (cần đăng ký) vào các trường của biến vừa khai báo
- Sử dụng hàm RegisterClass để đăng ký. Hàm cho giá trị TRUE nếu thành công và cho giá trị FALSE nếu không thành công.

+ Trước khi kết thúc chương trình thường dùng hàm UnregisterClass để huỷ bỏ lớp cửa sổ đã đăng ký.

Ví dụ về việc đăng ký một lớp cửa sổ:

```
WNDCLASS wc;
memset(&wc,0,sizeof(WNDCLASS));
// Thiết lập các thuộc tính của cửa sổ
wc.style= CS_HREDRAW |
          CS_VREDRAW |
          CS_BYTEALIGNWINDOW  ;
// Xác định hàm cửa sổ:  WndProc là tên hàm
   wc.lpfnWndProc = WndProc;
   wc.hInstance = hInst;    // Chỉ danh bản sao chương trình
   wc.lpszClassName = "DemoClass";  //Tên lớp cửa sổ
  // Tên menu (xác định trong tệp RC)
   wc.lpszMenuName = "DemoMenu";
   wc.cbClsExtra = 0;  // Thông tin bổ sung (không dùng)
   wc.cbWndExtra = 0; // Thông tin bổ sung (không dùng)
  // Quy định biểu tượng của chương trình
   wc.hIcon = LoadIcon(NULL,IDI_APPLICATION);
 // Xác định con trỏ của chương trình
 wc.hCursor = LoadCursor(NULL,IDC_ARROW);
// Xác định chổi tô vùng làm việc cửa sổ (mầu cửa sổ)
wc.hbrBackground=  (HBRUSH)GetStockObject(GRAY_BRUSH);
if(!RegisterClass(&wc))
```

```
    { // Thông báo lỗi khi đăng ký không thành
  MessageBox(NULL,"Loi DK lop",NULL,MB_ICONEXCLAMATION);
      return -1;
      }
       // Huỷ lớp cửa sổ đã đăng ký - sử dụng tên lớp
   UnregisterClass("DemoClass",hInst);
```

§3. XÂY DỰNG VÀ HIỂN THỊ CỬA SỔ

Sau khi thành công việc đăng ký cửa sổ, có thể xây dựng cửa sổ nhờ hàm CreateWindow:

Cú pháp của hàm:

HWND CreateWindow(LPCTSTR lpClassName, LPCTSTR lpWindowName, DWORD dwStyle, int x, int y, int nWidth, int nHeight, HWND hWndParent, HMENU hMenu, HINSTANCE hInstance, LPVOID lpParam)

Hàm CreateWindow dùng để xây dựng cửa sổ chính của chương trình theo một lớp đã đăng ký hoặc xây dựng cửa sổ con theo một lớp chuẩn của Windows (ví dụ lớp "Edit").

Các tham số của hàm: Hàm có 11 đối, ý nghĩa của chúng như sau:

+ lpClassName trỏ tới một chuỗi biểu thị tên lớp
+ lpWindowName trỏ tới một chuỗi biểu thị tiêu đề cửa sổ
+ dwStyle xác định các đặc tính của cửa sổ
+ x là hoành độ của điểm trên - trái của cửa sổ (đơn vị pixel) trên màn hình
+ y là tung độ của điểm trên-trái của cửa sổ trên màn hình
+ nWidth là độ rộng (theo chiều ngang) của cửa sổ
+ nHeight là độ cao (theo chiều dọc) của cửa sổ.
+ hWndParent là chỉ danh cửa sổ cha (bằng NULL nếu đây là cửa sổ chính) + hMenu là chỉ danh menu của cửa sổ (bằng NULL nếu dùng menu của lớp cửa sổ)
+ hInstance là chỉ danh bản sao chương trình.
+ lpParam trỏ đến các thông tin phụ khác (thường bằng NULL)

Giá trị của hàm

Nếu thành công hàm cho chỉ danh cửa sổ được xây dựng. Nếu không thành công, giá trị hàm bằng NULL

Ví dụ: Xây dựng cửa sổ theo lớp "DemoClass"

```
HWND hWndMain :  // Khai báo biến hWndMain để chứa chỉ danh cửa sổ
hWndMain= CreateWindow("DemoClass",
                  "Tinh dien tich cac hinh",
                  WS_CAPTION|WS_SYSMENU|WS_MINIMIZEBOX|
                  WS_MAXIMIZEBOX|
WS_THICKFRAME|WS_CLIPCHILDREN|WS_OVERLAPPED,
                  CW_USEDEFAULT,  // x mặc định
                  CW_USEDEFAULT,  // y mặc định
                  CW_USEDEFAULT,  // độ rộng mặc định
                  CW_USEDEFAULT,  // độ cao mặc định
                  NULL,  // không có cửa sổ cha
                  NULL,  // dùng menu của lớp "DemoClass"
                  hInst,  // chỉ danh bản sao chương trình
                 NULL);
  if(hWndMain= =NULL)
   {
    // Thông báo lỗi khi không thành công
      MessageBox(NULL,"Lỗi xây dựng cửa sổ",NULL,MB_OK);
      return -1;
   }
```

Hiển thị cửa sổ: Sau khi xây dựng cửa sổ bằng hàm CreateWindow thì cửa sổ vẫn chưa xuất hiện trên màn hình. Để hiển thị cửa sổ vừa xây dựng cần dùng hàm:

```
BOOL ShowWindow(HWND hWnd, int nCmdShow)
```

Trong đó:

+ hWnd là chỉ danh cửa sổ cần hiển thị
+ nCmdShow xác định dạng hiển thị (to, nhỏ,...)

Ví dụ: Để hiển thị cửa sổ có chỉ danh hWndMain theo dạng mặc định, có thể dùng câu lệnh sau:

```
ShowWindow(hWndMain, nCmdShow)
```

Chú ý:

+ Tham số nCmdShow là tham số thứ 4 của hàm WinMain
+ Cửa sổ sau khi hiện lên màn hình sẽ lập tức biến mất

+ Muốn cho cửa sổ hiển thị trong suốt thời gian hoạt động của chương trình thì phải sử dụng vòng lặp đợi nhận thông điệp:

```
MGS msg;
while(GetMessage(&msg,NULL,0,0))
 {
  TranslateMessage(&msg);
  DispatchMessage(&msg);
 }
```

Giải thích:

- Hàm GetMessage sẽ đón nhận thông điệp (từ hàng đợi chương trình) và gửi vào biến cấu trúc msg kiểu MSG.

Nếu thông điệp nhận được là WM_QUIT thì hàm cho giá trị 0 và thoát ra khỏi vòng lặp, kết thúc chương trình.

- Nếu có sự kiện bàn phím thì hàm TranslateMessage sẽ chuyển thành thông điệp tương ứng của Windows
- Hàm DispatchMessage sẽ gửi thông điệp này về Windows

§4. HÀM WINMAIN

Một chương trình C for Windows nhất thiết phải có hàm WinMain. Chương trình luôn luôn bắt đầu từ WinMain và cũng kết thúc tại WinMain. Hàm WinMain có cấu trúc nhất quán cho mọi chương trình. Khi viết một chương trình mới, có thể sử dụng hàm WinMain của chương trình cũ và sửa chữa vài chỗ.

4.1. Các đối của hàm và kiểu hàm:

Hàm có 4 đối:

+ Đối thứ nhất kiểu HINSTANCE chưa chỉ danh bản sao hiện tại của chương trình.

+ Đối thứ hai kiểu HINSTANCE chưa chỉ danh bản sao trước. Đối với lần chạy đầu tiên thì đối này nhận giá trị NULL.

+ Đối thứ ba kiểu LPSTR trỏ tới tham số dòng lệnh thực hiện chương trình từ chức năng RUN của Windows.

+ Đối thứ tư kiểu int xác định cách thức hiển thị cửa sổ chương trình.

Hàm có kiểu int WINAPI. Hàm có thể được khai báo như sau:

```
int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR lpszCmdLine,
                   int nCmdShow) ;
```

4.2. Nhiệm vụ của hàm WinMain

Hàm có các nhiệm vụ sau:

+ Lưu bản sao hiện tại của chương trình vào một biến toàn bộ.

+ Đăng ký lớp cửa sổ cho chương trình.

+ Xây dựng cửa sổ chính cho chương trình.

+ Hiển thị cửa sổ chính ra màn hình.

+ Xây dựng vòng lặp đợi, nhận thông điệp từ hàng đợi của chương trình.

+ Trước khi kết thúc chương trình cần huỷ lớp cửa sổ đã đăng ký.

4.3. Các biến và các kiểu dữ liệu dùng trong hàm WinMain

+ Các biến:

```
HINSTANCE hInst  ; // Dùng để lưu bản sao chương trình
HWND hWndMain ; // Dùng để lưu chỉ danh cửa sổ chính
WNDCLASS wc ; // Biến cấu trúc dùng để đăng ký lớp cửa sổ
MSG msg;    // Biến cấu trúc dùng để chứa nội dung thông điệp do Windows gửi tới.
```

4.4. Các hàm API dùng trong hàm WinMain

Hàm:

```
ATOM RegisterClass(CONST WNDCLASS *lpWndClass)
```

dùng để đăng ký một lớp cửa sổ theo các thuộc tính cho trong các thành phần một biến cấu trúc (ví dụ wc) do lpWndClass trỏ tới. Trước khi dùng hàm này cần dùng các lệnh gán để đưa các giá trị thích hợp vào các trường của wc. Hàm cho giá trị khác TRUE nếu đăng ký thành công và cho giá trị FALSE nếu đăng ký không thành công.

Hàm:

```
HWND CreateWindow(LPCTSTR lpClassName, LPCTSTR lpWindowName,
  DWORD dwStyle, int x, int y, int nWidth, int nHeight, HWND hWndParent,
  HMENU hMenu, HINSTANCE hInstance, LPVOID lpParam)
```

dùng để xây dựng cửa sổ chính của chương trình theo một lớp đã đăng ký hoặc xây dựng cửa sổ con theo một lớp chuẩn của Windows (ví dụ lớp "Edit"). Hàm này đã được giải thích bên trên.

Hàm

BOOL ShowWindow(HWND hWnd, int nCmdShow)

dùng để hiển thị cửa sổ có chỉ danh hWnd theo cách thức xác định bởi giá trị nguyên nCmdShow.

Hàm

GetMessage

dùng để nhận các thông điệp từ hàng đợi chương trình (do Windows gửi tới). Nguyên mẫu hàm như sau:

```
BOOL GetMessage(LPMSG lpMsg, HWND hWnd,
    UNIT wMsgFilterMin, UNIT wMsgFilterMax);
```

Thông điệp và các thông tin đi kèm được nhận và chứa trong một cấu trúc do con trỏ msg trỏ tới. Kiểu cấu trúc này được định nghĩa trong tệp windows.h như sau:

```
typedef  struct tagMSG
  {
    HWND hwnd ;  // Cửa sổ nhận thông điệp
    WORD  message ; // Thông điệp
    WORD  wParam ; // Tham số thứ nhất
    LONG  wParam ; // Tham số thứ hai
    DWORD time  ;  // thời gian gửi thông điệp tính bằng ms
    POINT   pt ;       // toạ độ x, y của chuột
  } MSG ;
```

Trường pt có kiểu POINT được định nghĩa trong windows.h như sau:

```
typedef  struct tagPOINT
   {
    LONG x, y;
   } POINT ;
```

Ý nghĩa các đối

lpMsg trỏ tới một biến kiểu MSG dùng để chứa thông điệp và các tham số bổ sung đi kèm

hWnd là chỉ danh cửa sổ liên quan đến thông điệp sẽ nhận. Một ứng dụng có thể có nhiều cửa sổ. Nếu muốn nhận tất cả các thông điệp của ứng dụng, thì cho hWnd bằng NULL.

Các đối wMsgFilterMin và wMsgFilterMax dùng để giới hạn miền giá trị của thông điệp sẽ nhận. Nếu chúng bằng 0 thì sẽ nhận được mọi thông điệp.

Hàm thường được sử dụng theo mẫu:

```
GetMessage(&msg,NULL,0,0);
```

trong đó msg là một biến cấu trúc kiểu MSG dùng để chứa nội dung thông điệp.

Hàm

```
BOOL TranslateMessage(COSNT MSG  *lpMsg)
```

dùng để chuyển mã phím ảo (khi bấm phím) thành các thông điệp ký tự và đặt vào một biến kiểu MSG (ví dụ msg) do lpMsg trỏ tới. Không phải tất cả các ứng dụng đều cần hàm này, nhưng đa số chương trình dùng chúng để làm việc với bàn phím.

Hàm

```
LONG DispatchMessage(COSNT MSG  *lpMsg)
```

dùng để chuyển thông điệp (chứa trong một biến kiểu MSG do lpMsg trỏ tới) cho Windows. Sau đó Windows sẽ chọn thời điểm thích hợp gửi thông điệp này cho chương trình xử lý.

Nhờ các hàm

TranslateMessage

DispatchMessage

mà chương trình có thể xử lý các sự kiện bàn phím.

Hàm

```
BOOL UnregisterClass(LPCTSTR  lpClassName, HINSTANCE hInstance)
```

dùng để huỷ lớp cửa sổ đã đăng ký. Hàm có 2 đối lần lượt xác định tên lớp cửa sổ cần huỷ và chỉ danh bản sao chương trình.

4.5. Ví dụ về hàm WinMain

```
char ten_ct[] = "vd1"; // Tên chương trình, tên lớp cửa sổ, đồng thời là tên
                       // menu mô tả trong tệp RC
HINSTANCE hInst; // Chứa chỉ danh bản sao hiện tại của chương trình
HWND hWndMain; // Chứa chỉ danh cửa sổ chính
int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE
hPrevInstance,
                   LPSTR lpszCmdLine,int nCmdShow)
```

```
{
  MSG msg;  WNDCLASS wc;
  hInst = hInstance;   //Lưu trữ chỉ danh bản sao chương trình
                       // Khởi gán cho các trường của biến wc
   memset(&wc, 0, sizeof(WNDCLASS));
    wc.style=CS_HREDRAW|CS_VREDRAW;
    wc.lpfnWndProc = WndProc;
    wc.hInstance = hInst;
    wc.lpszClassName = ten_ct;
    wc.lpszMenuName = ten_ct;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hIcon = LoadIcon(NULL,IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL,IDC_ARROW);
    wc.hbrBackground= GetStockObject(WHITE_BRUSH);
    if(!RegisterClass(&wc)) // Thông báo lỗi khi đăng ký không thành
     {
        MessageBeep(0xFFFFFFFF);
        MessageBox(NULL,"Loi DK lop",NULL,MB_OK);
        return 0;
     }
// Xây dựng cửa sổ chính và chứa chỉ danh trong biến hWndMain
 hWndMain= CreateWindow(ten_ct,
                  "Ve va tinh dien tich cac hinh",
                  WS_OVERLAPPEDWINDOW,
                 CW_USEDEFAULT,CW_USEDEFAULT,
                  CW_USEDEFAULT,CW_USEDEFAULT,
                  HWND_DESKTOP,
                  NULL,hInst,NULL);
   if(hWndMain==NULL) // Thông báo lỗi khi xây dựng không thành
    {
      MessageBox(NULL,"Loi tao CS",NULL,MB_OK);
      return 0;
    }
```

```
// Hiện cửa sổ
ShowWindow(hWndMain,nCmdShow);
        UpdateWindow(hWndMain);
  /* Vòng lặp nhận thông điệp
     Khai gặp thông điệp WM_QUIT  thì hàm GetMessage trả
     về giá trị 0 để thoát khỏi vòng lặp và kết thúc chương trình
  */
 while(GetMessage(&msg,NULL,0,0))
     {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
     }
 // Huỷ lớp cửa sổ đã đăng ký
  UnregisterClass(ten_ct,hInst);
  return msg.wParam;
}
```

Nhận xét: Khi chuyển sang một chương trình mới, thường chỉ cần sửa chữa 2 chỗ trong hàm WinMain, đó là:

+ Đặt tên mới cho chương trình, lớp cửa sổ và menu. Sửa câu lệnh đầu tiên:

```
char ten_ct[] = "NewName";
```

+ Đặt tiêu đề mới cho cửa sổ chính, bằng cách sửa tham số thứ 2 trong lời gọi hàm CreateWindow

4.6. Sự hoạt động của chương trình

Chương trình bắt đầu từ hàm WinMain để đăng ký lớp cửa sổ, xây dựng và hiển thị cửa sổ ra màn hình.

Nếu không có vòng lặp đợi nhận thông điệp thì cửa sổ sẽ lập tức biến mất khỏi màn hình.

Vòng lặp đợi nhận thông điệp có tác dụng sau:

+ Duy trì sự hiển diện của cửa sổ trong suốt thời gian hoạt động của chương trình.

+ Nhận các thông điệp có liên quan đến cửa sổ và chuyển cho hàm cửa sổ xử lý.

Như vậy vai trò của hàm WinMain là tổ chức một cửa sổ giao diện, còn toàn bộ việc giải quyết các chức năng chương trình nằm trong hàm cửa sổ và các hàm Dlg.

§5. SỰ KIỆN VÀ THÔNG ĐIỆP

Chương trình Windows hoạt động theo nguyên tắc: Đón nhận, phân loại và xử lý thông điệp. Vì vậy cần biết các nguồn gốc phát sinh ra thông điệp. Có thể phân thành các nhóm thông điệp sau:

+ Thông điệp liên quan đến menu (khi bấm vào một nút lệnh của menu)
+ Thông điệp liên quan đến cửa sổ:
  WM_CREATE
  WM_PAINT
  WM_CLOSE
  WM_SIZE
+ Thông điệp bàn phím
+ Thông điệp chuột
+ Thông điệp thời gian

Chú ý cần phân biệt sự kiện và thông điệp. Sự kiện thường dùng để diễn tả một thao tác của người sử dụng, như chọn một nút lệnh, gõ bàn phím hoặc bấm một phím chuột. Các thao tác đó sẽ được Windows nhận và mã hoá thành thông điệp chuẩn gồm 3 giá trị là Message, wParam và lParam. Dựa vào các giá trị này, mà chúng ta viết các câu lệnh để xử lý sự kiện.

§6. HÀM CỬA SỔ

6.1. Các đối của hàm và kiểu hàm:

Hàm có 4 đối:

+ Đối thứ nhất kiểu HWND chưa chỉ danh cửa sổ.

+ Đối thứ hai kiểu UINT chưa thông điệp của Windows .

+ Đối thứ ba kiểu WPARAM chứa tham số thứ nhất kiểu nguyên không dấu được gửi cùng thông điệp.

+ Đối thứ tư kiểu LPARAM chứa tham số thứ hai kiểu nguyên dài được gửi cùng thông điệp.

Hàm có kiểu LRESULT CALLBACK

Tên hàm do người lập trình tự chọn, dưới đây ta thường dùng tên WndProc. Hàm có thể khai báo như sau:

```
LRESULT CALLBACK  WndProc(HWND hWnd, UINT  Message,
        WPARAM   wParam, LPARAM   lParam);
```

6.2. Nhiệm vụ của hàm WndProc

6.2.1. Hàm WndProc được Windows gọi để xử lý các thông điệp có liên quan tới các sự kiện xẩy ra trên cửa sổ chính và các thông điệp khác của hệ thống (Hàm WndProc không có lời gọi trong chương trình). Windows sẽ gửi 4 giá trị cho 4 đối của hàm như sau:

+ Đối hWnd sẽ nhận chỉ danh cửa sổ chính.

+ Đối Message nhận thông điệp của Windows. Tên thông điệp Windows có 2 ký tự đầu là WM và được định nghĩa trong tệp Windows.h dưới dạng một số nguyên không dấu.

+ Đối wParam sẽ nhận tham số thứ nhất

+ Đối lParam sẽ nhận tham số thứ hai

Các giá trị Message, wParam và lParam sẽ cung cấp đầy đủ thông tin về một sự kiện nào đó mà Windows cung cấp cho hàm WndProc. Trong 3 giá trị này, thì Message là quan trọng nhất, kế đó là wParam, cuối cùng là lParam. Hàm WndProc sẽ dựa vào các gía trị này để biết sự kiện gì đã xẩy ra và cần xử lý ra sao.

6.2.2. Vài ví dụ về thông điệp và các tham số

+ Khi bấm chuột tại một nút lệnh (trên hệ menu của cửa sổ chính) thì Windows sẽ gọi hàm WndProc và sẽ gán cho các đối của hàm các giá trị sau:

- Đối Message nhận thông điệp WM_COMMAND
- Đối wParam nhận định danh nút lệnh được chọn

Để cho gọn, sau này sẽ nói như sau: Khi bấm chuột tại một nút lệnh thì

Message = WM_COMMAND

wParam = Định danh nút lệnh

+ Khi nhấn một phím ký tự (trong phạm vi cửa sổ chính) thì :

Message = WM_CHAR

wParam = mã ASCII của ký tự ứng với phím được bấm

+ Khi nhấn một phím chức năng (trong phạm vi cửa sổ chính) thì:

Message = WM_KEYDOWN

wParam = mã phím ảo ứng với phím được ấn

+ Khi nhấn nút trái chuột (trong phạm vi cửa sổ chính) thì:

Message = WM_LBUTTONDOWN

LOWORD(lParam) = vị trí cột của con trỏ chuột

HIWORD(lParam) = Vị trí hàng của con trỏ chuột

6.3. Các biến thường khai báo và dùng trong hàm WndProc

```
HDC  hDC ;  // Để nhận chỉ danh ngữ cảnh của cửa sổ
            // dùng trong hàm xuất dữ liệu trên cửa sổ
PAINSTRUCT  ps ; // Dùng trong thông điệp WM_PAINT
RECT rect ;  // Dùng để xác định vùng làm việc cửa sổ
```

6.4. Các hàm API dùng trong hàm WndProc

Hàm:

DialogBox

dùng để hiển thị hộp thoại.

Hàm:

PostMessage

dùng để gửi đi một thông điệp. Hàm WndProc sẽ nhận và xử lý thông điệp này.

Hàm:

PostQuitMessage

dùng để gửi thông điệp WM_QUIT để kết thúc chương trình.

Hàm:

DefWindowProc

dùng để nhận các thông điệp mà ta không muốn xử lý trong các hàm cửa sổ và các hàm DLG.

6.5. Ví dụ về hàm WndProc

```
LRESULT CALLBACK WndProc(HWND hWnd,UINT Message,
WPARAM wParam, LPARAM lParam)
{
  HDC hDC;
  PAINTSTRUCT ps;
  RECT rect;
  switch(Message)
   {
    case WM_COMMAND:  // Xử lý các nút lệnh của menu
     switch(wParam)
      {
       case IDM_TG:  //Xử lý nút lệnh IDM_TG
```

```
            DialogBox(hInst,"TGDlg",hWnd,TGDlgProc);
            break;
        case IDM_HT:  //Xử lý nút lệnh IDM_HT
            DialogBox(hInst,"HTDlg",hWnd,HTDlgProc);
            break;
        case IDM_VE_ELIP:    //Xử lý nút lệnh IDM_VE_ELIP
            hDC=GetDC(hWnd);
            GetClientRect(hWnd,&rect);
            Ellipse(hDC,20,20,rect.right-20,rect.bottom-20);
            TextOut(hDC,rect.right/2,rect.bottom/2,"HELLO",5);
            ReleaseDC(hWnd,hDC);
            break;
        case IDM_VE_CN:  //Xử lý nút lệnh IDM_VE_CN
            ve_cn = TRUE;
            InvalidateRect(hWnd,NULL,TRUE);
            break;
        case IDM_XOA: //Xử lý nút lệnh IDM_XOA
            ve_cn= FALSE;
            InvalidateRect(hWnd,NULL,TRUE);
            break;
        case IDM_KT: //Xử lý nút lệnh IDM_KT
            PostMessage(hWnd,WM_CLOSE,0,0L);
            break;
        default:  // Bỏ qua các thông điệp khác về nút lệnh
            return DefWindowProc(hWnd,Message,wParam,lParam);
            break;
      }
    break;
  case WM_PAINT:  //Xử lý thông điệp WM_PAINT
    memset(&ps,0x00,sizeof(PAINTSTRUCT));
    hDC = BeginPaint(hWnd,&ps);
    if(ve_cn)
        {
         GetClientRect(hWnd,&rect);
```

§5. SỰ KIỆN VÀ THÔNG ĐIỆP

Chương trình Windows hoạt động theo nguyên tắc: Đón nhận, phân loại và xử lý thông điệp. Vì vậy cần biết các nguồn gốc phát sinh ra thông điệp. Có thể phân thành các nhóm thông điệp sau:

+ Thông điệp liên quan đến menu (khi bấm vào một nút lệnh của menu)

+ Thông điệp liên quan đến cửa sổ:

WM_CREATE

WM_PAINT

WM_CLOSE

WM_SIZE

+ Thông điệp bàn phím

+ Thông điệp chuột

+ Thông điệp thời gian

Chú ý cần phân biệt sự kiện và thông điệp. Sự kiện thường dùng để diễn tả một thao tác của người sử dụng, như chọn một nút lệnh, gõ bàn phím hoặc bấm một phím chuột. Các thao tác đó sẽ được Windows nhận và mã hoá thành thông điệp chuẩn gồm 3 giá trị là Message, wParam và lParam. Dựa vào các giá trị này, mà chúng ta viết các câu lệnh để xử lý sự kiện.

§6. HÀM CỬA SỔ

6.1. Các đối của hàm và kiểu hàm:

Hàm có 4 đối:

+ Đối thứ nhất kiểu HWND chứa chỉ danh cửa sổ.

+ Đối thứ hai kiểu UINT chứa thông điệp của Windows.

+ Đối thứ ba kiểu WPARAM chứa tham số thứ nhất kiểu nguyên không dấu được gửi cùng thông điệp.

+ Đối thứ tư kiểu LPARAM chứa tham số thứ hai kiểu nguyên dài được gửi cùng thông điệp.

Hàm có kiểu LRESULT CALLBACK

Tên hàm do người lập trình tự chọn, dưới đây ta thường dùng tên WndProc. Hàm có thể khai báo như sau:

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT Message,
    WPARAM wParam, LPARAM lParam);
```

```
        DialogBox(hInst,"TGDlg",hWnd,TGDlgProc);
        break;
      case IDM_HT:  //Xử lý nút lệnh IDM_HT
        DialogBox(hInst,"HTDlg",hWnd,HTDlgProc);
        break;
      case IDM_VE_ELIP:    //Xử lý nút lệnh IDM_VE_ELIP
        hDC=GetDC(hWnd);
        GetClientRect(hWnd,&rect);
        Ellipse(hDC,20,20,rect.right-20,rect.bottom-20);
        TextOut(hDC,rect.right/2,rect.bottom/2,"HELLO",5);
        ReleaseDC(hWnd,hDC);
        break;
      case IDM_VE_CN:  //Xử lý nút lệnh IDM_VE_CN
        ve_cn = TRUE;
        InvalidateRect(hWnd,NULL,TRUE);
        break;
      case IDM_XOA: //Xử lý nút lệnh IDM_XOA
        ve_cn= FALSE;
        InvalidateRect(hWnd,NULL,TRUE);
        break;
      case IDM_KT: //Xử lý nút lệnh IDM_KT
        PostMessage(hWnd,WM_CLOSE,0,0L);
        break;
      default:  // Bỏ qua các thông điệp khác về nút lệnh
        return DefWindowProc(hWnd,Message,wParam,lParam);
        break;
    }
    break;
  case WM_PAINT:  //Xử lý thông điệp WM_PAINT
    memset(&ps,0x00,sizeof(PAINTSTRUCT));
    hDC = BeginPaint(hWnd,&ps);
    if(ve_cn)
      {
        GetClientRect(hWnd,&rect);
```

```
            Rectangle(hDC,20,20,rect.right-20,rect.bottom-20);
           }
          EndPaint(hWnd,&ps);
         break ;
       case WM_CLOSE: //Xử lý thông điệp WM_CLOSE
         DestroyWindow(hWnd);
         PostQuitMessage(0);
         break ;
       default: //bỏ qua các thông điệp khác
          return DefWindowProc(hWnd,Message,wParam,lParam);
    }
  return 0L;
}
```

6.6. Cách trình bầy hàm WndProc

Cấu trúc hàm WndProc gồm 3 phần:

+ Khai báo các biến dùng trong hàm
+ Dùng lệnh switch theo wParam để xử lý các nút lệnh
+ Dùng lệnh switch theo Message để xử lý các thông điệp khác

Vì vậy để ngắn gọn và rõ nghĩa, dưới đây chúng ta sẽ trình bầy hàm WndProc theo 3 phần như trên. Ví dụ đối với hàm WndProc trong 6.5 có thể trình bầy như sau:

Phần 1. Khai báo biến

```
HDC hDC;
PAINTSTRUCT ps;
RECT rect;
```

Phần 2. Xử lý các nút lệnh

```
case IDM_TG:  //Xử lý nút lệnh IDM_TG
   DialogBox(hInst,"TGDlg",hWnd,TGDlgProc);
   break;
case IDM_HT:  //Xử lý nút lệnh IDM_HT
        DialogBox(hInst,"HTDlg",hWnd,HTDlgProc);
      break;
case IDM_VE_ELIP:   //Xử lý nút lệnh IDM_VE_ELIP
```

```
            hDC=GetDC(hWnd);
             GetClientRect(hWnd,&rect);
             Ellipse(hDC,20,20,rect.right-20,rect.bottom-20);
             TextOut(hDC,rect.right/2,rect.bottom/2,"HELLO",5);
             ReleaseDC(hWnd,hDC);
            break;
        case IDM_VE_CN:  //Xử lý nút lệnh IDM_VE_CN
            ve_cn = TRUE;
             InvalidateRect(hWnd,NULL,TRUE);
            break;
        case IDM_XOA: //Xử lý nút lệnh IDM_XOA
             ve_cn= FALSE;
             InvalidateRect(hWnd,NULL,TRUE);
             break;
        case IDM_KT: //Xử lý nút lệnh IDM_KT
             PostMessage(hWnd,WM_CLOSE,0,0L);
             break;
```

Phần 3. Xử lý các thông điệp khác

```
    case WM_PAINT:
        //Xử lý thông điệp WM_PAINT
      memset(&ps,0x00,sizeof(PAINTSTRUCT));
      hDC = BeginPaint(hWnd,&ps);
      if(ve_cn)
          {
           GetClientRect(hWnd,&rect);
          Rectangle(hDC,20,20,rect.right-20,rect.bottom-20);
          }
       EndPaint(hWnd,&ps);
      break ;
    case WM_CLOSE:
       //Xử lý thông điệp WM_CLOSE
      DestroyWindow(hWnd);
      PostQuitMessage(0);
      break ;
```

§7. CỬA SỔ CON SOẠN THẢO

7.1. Khái niệm cửa sổ con soạn thảo

+ Trên vùng làm việc cửa sổ chính cho phép hiển thị dữ liệu văn bản và đồ hoạ, nhưng không cho phép soạn thảo dữ liệu.

+ Để soạn thảo dữ liệu có thể dùng cửa sổ con soạn thảo hoặc hộp hội thoại.

+ Cách xây dựng và hiển thị cửa sổ con soạn thảo (trên vùng làm việc của cửa sổ chính) như sau:

```
RECT r;
hWndEdit
GetClientRect(hWnd,&r); // Luon luon (r.left,r.top) = (0,0)
hWndEdit = CreateWindow("edit","CS 1",
    WS_CHILD | WS_VISIBLE | WS_HSCROLL | WS_VSCROLL |
WS_BORDER | ES_LEFT | ES_MULTILINE | ES_NOHIDESEL|
ES_AUTOHSCROLL | ES_AUTOVSCROLL,  0,0,(r.right-r.left)/2,
(r.bottom-r.top)/2, hWnd, (HMENU)1,hInst,NULL);
```

Chú ý:

Hàm WndProc (xem §6) không nhận được các sự kiện chuột và bàn phím trên cửa sổ con soạn thảo.

7.2. Các hàm dùng cho cửa sổ con soạn thảo

1. Nhận văn bản của CS con soạn thảo và chứa vào bộ nhớ (xem §8)

int GetWindowText(HWND hWnd, LPSTR lpString,int nMaxCount)

2. Đưa văn bản từ bộ nhớ ra CS con soạn thảo (xem §8)

BOOL SetWindowText(HWND hWnd, LPSTR lpString)

3. Xác định độ dài của văn bản chứa trong CS con soạn thảo (xem §8)

int GetWindowTextLength(HWND hWnd)

4. Xác định lại kích thước của CS con soạn thảo (xem §8)

BOOL MoveWindow(HWND hWnd, int x, int y,
int nWidth, int nHeight, BOOL bRepaint)

5. Gửi thông điệp đến CS con soạn thảo

LRESULT SendMessage (HWND hWnd, UINT Message,
WPARAM wParam, LPARAM lParam)

Tuỳ theo giá trị của các tham số, mà hàm cho những kết quả khác nhau. Dưới đây là một số trường hợp hay dùng đối với CS con soạn thảo.

TT	Message	wParam	lParam	Giá hàm, công dụng hàm
1	WM_UNDO	0	0	Undo thao tác cuối cùng
2	WM_CLEAR	0	0	Bỏ đánh dấu
3	WM_COPY	0	0	Sao khối văn bản chọn vào Clipboard
4	WM_CUT	0	0	Cất khối văn bản chọn vào Clipboard
5	WM_PASTE	0	0	Sao từ Clipboard vào CS con soạn thảo
6	EM_LIMITTEXT	nMax	0	Quy định số ký tự tối đa của CS soạn thảo là nMax, nếu nMax=0 thì hiểu là 0XFFFFFFFF
7	EM_GETLINECOUNT	0	0	Giá trị hàm là số dòng trong CS soạn thảo
8	EM_GETLINE	n	lpBuf	Sao dòng thứ n (n=0, 1, ..) của CS soạn thảo vào lpBuf. Giá trị hàm bằng số ký tự của dòng n

7.3. Ví dụ về cửa sổ con soạn thảo

Chương trình soạn thảo dưới đây gồm các chức năng: Mở một tệp (khi chưa tồn tại thì tạo mới), ghi văn bản đang soạn thảo lên tệp. Chương trình minh hoạ các vấn đề sau:

+ Xây dựng cửa sổ con soạn thảo
+ Lấy văn bản từ CS con soạn thảo đưa vào mảng
+ Đưa văn bản trong mảng ra CS con soạn thảo
+ Hộp thoại chuẩn OpenFile
+ Đọc, ghi tệp
+ Cấp phát bộ nhớ

Chương trình trong: **D:\WVC6\C-WIN\Chg2\St_tep_2**

//Tệp Resource.rc

```
#include "resource.h"
#include "windows.h"
S_THAO MENU
 BEGIN
  POPUP  "File"
   BEGIN
    MENUITEM "Open", IDM_F_OPEN
    MENUITEM "Save", IDM_F_SAVE, GRAYED
   END
  MENUITEM "Exit", IDM_EXIT
END
```

//Tệp Resource.h

```
#define IDM_FILE          1000
#define IDM_F_OPEN          1050
#define IDM_F_SAVE          1100
#define IDM_EXIT          2000
```

//Tệp S_thao.c

```
/*
D:\WVC6\C-WIN\Chg2\St_tep_2\S_thao.c
Minh hoa:
   + Cua so con soan thao
   + Lay van ban tu CS con soan thao dua vao mang
   + Dua van ban tu mang ra CS con soan thao
   + Hop thoai chuan OpenFile
   + Doc, ghi tep
   + Cap phat bo nho
*/
#include "resource.h"
#include "windows.h"
#include "commdlg.h"
```

```
#include "malloc.h"
#include "stdio.h"
#include "string.h"
#define IDC_EDIT 3000
long FileLength(FILE*);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
char szAppName[20];
HINSTANCE hInst;
HWND hWndMain;
HWND hWndEdit;
RECT rr;
OPENFILENAME ofn;
char filename[125], titlename[125];
char filter[]="C file(*.c)\0*.c\0H File\
(*.h)\0*.h\0All File(*.*)\0*.*\0";
FILE *fp;
long nn;
PSTR pEditBuf;
char tb[40];
int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,
                                    LPSTR lpszCmdLine, int nCmdShow)
{
 MSG       msg;
 WNDCLASS  wndclass;
 strcpy(szAppName, "S_THAO");
 hInst = hInstance;
 memset(&wndclass, 0x00, sizeof(WNDCLASS));
 wndclass.style = CS_HREDRAW | CS_VREDRAW |
CS_BYTEALIGNWINDOW;
 wndclass.lpfnWndProc = WndProc;
 wndclass.cbClsExtra = 0;
 wndclass.cbWndExtra = 0;
 wndclass.hInstance = hInst;
 wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
```

```
wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
wndclass.hbrBackground = GetStockObject(WHITE_BRUSH);
wndclass.lpszMenuName = szAppName;  /* Menu Name is App Name */
wndclass.lpszClassName = szAppName;  /* Class Name is App Name */
if(!RegisterClass(&wndclass))
{
  MessageBox(NULL,"Loi DK Lop CS",NULL,MB_OK);
      return 0;
}
hWndMain = CreateWindow(
        szAppName,
        "Chuong trinh soan thao",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        HWND_DESKTOP,
        NULL,
        hInst,
        NULL);
if(hWndMain == NULL)
 {
   MessageBox(NULL, "Loi XD CS", NULL, B_ICONEXCLAMATION);
    return 0;
 }
GetClientRect(hWndMain,&rr);
 hWndEdit = CreateWindow("Edit",
     NULL,
              WS_CHILD | WS_VISIBLE |WS_VSCROLL |
              WS_HSCROLL | WS_BORDER |
              ES_AUTOVSCROLL | ES_AUTOHSCROLL |
              ES_LEFT |ES_NOHIDESEL | ES_MULTILINE ,0,0,
              rr.right - rr.left,   rr.bottom - rr.top,  hWndMain,
             (HMENU)IDC_EDIT,  hInst,  NULL);
 ShowWindow(hWndMain, nCmdShow);
```

```
 SetFocus(hWndEdit);
 while(GetMessage(&msg, NULL, 0, 0))
  {
   TranslateMessage(&msg);
   DispatchMessage(&msg);
  }
 return msg.wParam;
}
LRESULT CALLBACK WndProc(HWND hWnd, UINT Message,
                         WPARAM wParam, LPARAM lParam)
{
 HMENU hMenu;
 switch (Message)
  {
  case WM_COMMAND:
      switch (wParam)
       {
        case IDM_F_OPEN:
            GetOpenFileName(&ofn);
            SetWindowText(hWnd,filename);
            fp= fopen(filename,"rb")   ;
            if(fp!=NULL) //Ton tai
             {
              long m;
              nn= FileLength(fp);

              pEditBuf = (PSTR)malloc(nn+1);
              if(pEditBuf==NULL)
                {
                 MessageBox(NULL,"Loi cap BN",NULL,MB_OK);
                }
              m=fread(pEditBuf,1,nn,fp);
              if(m!=nn)
                {
```

```
            MessageBox(NULL,"Loi doc",NULL,MB_OK);
          }
        pEditBuf[nn]='\0';
        SetWindowText(hWndEdit,pEditBuf);
        fclose(fp);
        free(pEditBuf);
        }
      else // Tep chua ton tai
        {
          SetWindowText(hWndEdit," ");
        }
      hMenu=GetMenu(hWnd);
      EnableMenuItem(hMenu,IDM_F_SAVE,MF_ENABLED);
      SetFocus(hWndEdit);
      break;
  case IDM_F_SAVE:
    {
      long m;
      nn= GetWindowTextLength(hWndEdit);
      pEditBuf = (PSTR)malloc(nn+1);
      GetWindowText(hWndEdit,pEditBuf,nn+1);
      fp= fopen(filename,"wb")   ;
      m=fwrite(pEditBuf,1,nn,fp);
      if(m!=nn)
        MessageBox(NULL,"loi ghi",NULL,MB_OK);
      fclose(fp);
      free(pEditBuf);
    }
    break;
  case IDM_EXIT:
      PostMessage(hWnd,WM_CLOSE,0,0);
      break;
  default:
    return DefWindowProc(hWnd, Message, wParam, lParam);
```

```
        }
        break;
    case WM_CREATE:
        ofn.lStructSize=sizeof(OPENFILENAME);
        ofn.hwndOwner=hWnd;
        ofn.lpstrFilter=filter;
        ofn.lpstrFile=filename;
        ofn.nMaxFile=125;
        ofn.lpstrFileTitle=titlename;
        ofn.nMaxFileTitle=125;
        ofn.lpstrDefExt="c";
        break;
    case WM_SIZE:
MoveWindow(hWndEdit,0,0,LOWORD(lParam),HIWORD(lParam),FALSE);
        break;
    case WM_CLOSE:
        DestroyWindow(hWnd);
        if (hWnd == hWndMain)
          PostQuitMessage(0);
       break;
    default:
        return DefWindowProc(hWnd, Message, wParam, lParam);
   }
 return 0L;
}
long FileLength(FILE *fp)
{
 long vtht,dd;
 vtht = ftell(fp);
 fseek(fp,0,SEEK_END);
 dd=ftell(fp);
 fseek(fp,vtht,SEEK_SET);
 return dd;
}
```

§8. TÓM TẮT MỘT SỐ HÀM VỀ CỬA SỔ

1. Hàm

BOLL CloseWindow(HWND hWnd);

+ Công dụng: Làm cực tiểu cửa sổ

+ Đối

hWnd là chỉ danh cửa sổ cần làm cực tiểu.

+ Giá trị trả về

Hàm có giá trị TRUE (khác 0) nếu thành công và có giá trị FALSE (bằng 0) nếu có lỗi.

+ Chú ý:

Có thể khôi phục cửa sổ bằng câu lệnh:

ShowWindow(hWnd, SW_RESTORE);

2. Hàm

CreateWindow (xem bên trên)

3. Hàm

LRESULT DefWindowProc(HWND hWnd, UINT Message, WPARAM wParam, LPARAM lParam)

+ Công dụng: Gọi thủ tục cửa sổ mặc định để xử lý các thông điệp mà chương trình bỏ qua

4. Hàm

BOOL DestroyWindow(HWND hWnd)

+ Công dụng: Dùng để huỷ cửa sổ
+ Đối

hWnd là chỉ danh cửa sổ cần huỷ (phá bỏ)

+ Giá trị trả về

Hàm có giá trị TRUE (khác 0) nếu thành công và có giá trị FALSE (bằng 0) nếu có lỗi.

5. Hàm

BOOL EnableWindow(HWND hWnd, BOOL bEnable)

+ Công dụng: Để cho hoặc không cho cửa sổ (hay ô điều khiển) nhận tín hiệu nhập dữ liệu từ chuột và bàn phím

+ Các đối:

hWnd là chỉ danh cửa sổ hay chỉ danh ô điều khiển

bEnable bằng TRUE để cho phép, bằng FALSE để không cho phép nhận tín hiệu nhập từ chuột và bàn phím.

6. Hàm

GetClientRect(HWND hWnd, LPRECT lpRect)

+Công dụng: Nhận toạ độ vùng làm việc của cửa sổ

+ Các đối:

hWnd là chỉ danh cửa sổ

lpRect chứa địa chỉ của một biến cấu trúc kiểu RECT (xem bên dưới) dùng để chứa toạ độ của điểm trên-trái và dưới-phải của vùng làm việc cửa sổ. Cụ thể:

Hoành độ, tung độ của điểm trên-trái chứa trong các thành phần left và top

Hoành độ, tung độ của điểm dưới-phải chứa trong các thành phần right và bottom

Kiểu RECT được định nghiã trong windows.h như sau:

```
typedef struct _RECT
  {
     LONG left, top, right, bottom;
  } RECT;
```

+ Chú ý: Vì đây là toạ độ tương đối trong vùng làm việc cửa sổ nên:

left = 0 và top = 0

7. Hàm

int GetWindowText(HWND hWnd, LPSTR lpString, int nMaxCount)

+ Công dụng: Để sao chép tiêu đề cửa sổ chính hoặc văn bản của cửa sổ con soạn thảo vào một vùng nhớ.

+ Các đối:

- hWnd là chỉ danh cửa sổ
- lpString trỏ tới vùng nhớ dùng để chứa văn bản sao chép
- nMaxCount là số ký tự cực đại cần sao chép, kể cả ký tự kết thúc (có mã 0)

+ Giá trị của hàm:

Nếu thành công giá trị trả về bằng độ dài (không kể ký tự kết thúc) của văn bản sao chép được

Nếu có lỗi giá trị trả về bằng 0

8. Hàm

int GetWindowTextLength(HWND hWnd)

+ Công dụng: Hàm cho biết độ dài của tiêu đề cửa sổ chính hoặc độ dài của văn bản chứa trong cửa sổ con soạn thảo.

+ Đối hWnd là chỉ danh cửa sổ

9. Hàm

BOOL MoveWindow(HWND hWnd, int x, int y,
int nWidth, int nHeight, BOOL bRepaint)

+ Công dụng: Xác định lại vị trí và kích thước của cửa sổ. Đối với cửa sổ chính vị trí xác định theo góc trên-trái màn hình, đối với cửa sổ con theo góc trên-trái của cửa sổ chính

+ Các đối:

- hWnd là chỉ danh cửa sổ cần xác định lại vị trí và kích thước
- x, y là hoành độ và tung độ mới của góc trên-trái của cửa sổ hWnd
- nWidth, nHeight là chiều rộng và chiều cao mới của cửa sổ hWnd
- bRepaint cho biết cửa sổ có bị vẽ lại không. Nếu bRepaint bằng TRUE thì cửa sổ sẽ được vẽ lại ngay sau khi dịch chuyển đến vị trí mới.

10. Hàm

RegisterWindowClass

dùng để đăng ký một lớp cửa sổ (xem bên trên)

11. Hàm

LRESULT SendMessage (HWND hWnd, UINT Message,
WPARAM wParam, LPARAM lParam)

+ Công dụng: Gửi thông điệp đến hàm cửa sổ và yêu cầu xử lý ngay (xem chi tiết trong chương Hộp hội thoại)

12. Hàm

HWND SetFocus(HWND hWnd)

+ Công dụng: Đặt focus bàn phím cho cửa sổ

+ Đối:

hWnd là chỉ danh cửa sổ nhận focus bàn phím

+ Giá trị hàm là chỉ danh cửa sổ trước đó có focus bàn phím

13. Hàm

BOOL SetWindowText(HWND hWnd, LPSTR lpString)

+ Công dụng: Thay đổi tiêu đề của cửa sổ chính hay văn bản của cửa sổ con soạn thảo

+ Các đối:

- hWnd là chỉ danh cửa sổ

- lpString trỏ tới chuỗi được dùng làm nội dung mới cho cửa sổ hWnd

+ Giá trị của hàm bằng TRUE (khác không) nếu thành công.

14. Hàm

BOOL ShowWindow(HWND hWnd, int nCmdShow)

+ Công dụng: Ẩn hoặc hiện cửa sổ

+ Các đối:

- hWnd là chỉ danh cửa sổ

- nCmdShow quy định trạng thái ẩn, hiện cửa sổ. Sau đây là một vài giá trị nCmdShow:

SW_HIDE làm ẩn cửa sổ

SW_MAXIMIZE làm cực đại kích cỡ cửa sổ

SW_MINIMIZE làm cực tiểu kích cỡ cửa sổ

SW_RESTORE khôi phục kích thước trước khi cực đại, cực tiểu

SW_SHOW hiển thị cửa sổ lên màn hình

15. Hàm

BOOL UnregisterClass(LPSTR lpClassName, HISTANCE hInstance)

+ Công dụng: Huỷ lớp cửa sổ và giải phóng vùng nhớ của lớp

+ Các đối:

- lpClassName là địa chỉ của chuỗi dùng làm tên lớp

- hInstance là chỉ danh bản sao chương trình

16. Hàm

BOOL UpdateWindow(HWND hWnd)

+ Công dụng: Update vùng làm việc cửa sổ bằng cách gửi thông điệp WM_PAINT trực tiếp đến hàm cửa sổ

+ Đối:

hWnd là chỉ danh cửa sổ cần update

CHƯƠNG 3

MENU

§1. THIẾT KẾ MENU

Menu được thiết kế trong tệp tài nguyên rc theo mẫu sau:

```
MenuName   MENU  [Option]
  BEGIN
     Các mục menu
  END
```

Trong đó:

+ MenuName là tên của menu (có thể là số nguyên). Thông thường tên menu trùng với tên chương trình.

+ MENU là từ khoá

+ Option có thể là các lựa chọn sau:

DISCARDABLE - Có thể loại bỏ menu để dành bộ nhớ cho ứng dụng khác, menu được tự động nạp lại khi cần.

LOADONCALL - Menu được nạp vào khi cần

FIXED - Menu cố định trong bộ nhớ

MOVEABLE - Menu có thể dịch chuyển trong bộ nhớ và không có địa chỉ cố định

PRELOAD - Nạp menu khi chương trình bắt đầu chạy

Các macro này được định nghĩa trong tệp windows.h

+ Các từ khoá BEGIN và END có thể thay bằng các dấu ngoặc nhọn { và }

+ Có 2 kiểu mục menu là MENUITEM và POPUP:

MENUITEM là một nút lệnh hoặc dấu phân cách

POPUP là một menu con, nó lại bao gồm các MENUITEM và POPUP khác. Khuôn mẫu tổng quát của chúng như sau:

```
MENUITEM  "&ItemName", MenuID [, option]
MENUITEM  SEPARATOR
POPUP  "&PopupName" [, option]
  BEGIN
```

Các mục menu

END

Trong đó:

+ ItemName là tên của nút lệnh, ví dụ Open, Save,... Ký tự sau dấu & của ItemName sẽ được gạch chân, gọi là phím lệnh. Nó kết hợp với phím Alt để chọn nút lệnh. Phím lệnh có thể là ký tự bất kỳ trong ItemName, sao cho không bị trùng lặp với các lệnh khác cùng mức.

+ MenuID là định danh của nút lệnh. Định danh của các nút lệnh trong menu cần không được trùng nhau. Các định danh được định nghĩa bằng câu lệnh #define (trong tệp tiêu đề đuôi h) dưới dạng một số nguyên không dấu.

+ option có thể là các macro sau:

CHECKED - Đánh dấu mục menu (chỉ dùng cho nút lệnh)

GRAYED - Mục menu bị mờ và không cho chọn

INACTIVE - Không cho chọn mục menu nhưng vẫn hiển thị bình thường

+ MENUITEM SEPARATOR - vẽ đường nằm ngang để phân cách các mục menu

Ví dụ

```
MyMenu  MENU  DISCARDABLE
  {
    POPUP  "&File"
      {
        MENUITEM "&New" , IDM_F_NEW , CHECKED
        MENUITEM "&Open" , IDM_F_OPEN
        MENUITEM SEPARATOR
        MENUITEM "&Save" , IDM_F_SAVE , GRAYED
        MENUITEM  "Save &As" , IDM_F_SAVEAS, GRAYED
      }
     POPUP  "&Edit" , GRAYED
       {
        MENUITEM "&Copy" , IDM_E_COPY
        MENUITEM " C&ut" , IDM_E_CUT
        MENUITEM "&Paste" , IDM_E_PASTE , GRAYED
        }
      MENUITEM "E&xit", IDM_EXIT
   }
```

§2. GẮN MENU VÀO CỬA SỔ

Dưới đây sẽ trình bầy 2 cách để gắn menu vào cửa sổ.

Cách 1

Khai báo một biến cấu trúc kiểu WNDCLASS , ví dụ:

```
WNDCLASS  wc;
```

Gán tên menu vào trường lpszMenuName của biến wc:

```
wc. lpszMenuName = "MyMenu"
```

Gán các giá trị thích hợp vào các trường khác của wc

Dùng wc để đăng ký lớp cửa sổ:

```
RegisterClass(&wc);
```

Dùng tên lớp để tạo cửa sổ chính

Chú ý: Nếu tên menu là một số nguyên, ví dụ 100 , thì phép gán tên menu sẽ như sau:

```
wc. lpszMenuName = MAKEINTRESOURCE(100) ;
```

Cách 2

Gán NULL vào trường lpszMenuName:

```
wc. lpszMenuName = NULL;
```

Tạo cửa sổ chính, khi đó cửa sổ chưa có menu

Nạp menu bằng các câu lệnh:

```
HMENU  hMenu ;
hMenu = LoadMenu(hInst , "MyMenu");
```

Gắn menu với cửa sổ:

```
SetMenu(hWnd, hMenu);
```

Chú ý: Nếu tên menu là số nguyên, ví dụ 100, thì câu lệnh nạp menu được viết như sau:

```
hMenu = LoadMenu(hInst ,MAKEINTRESOURCE(100));
```

Có thể dùng cách thứ 2 để thay đổi hệ menu của cửa sổ chính như sau:

```
HMENU  hOldMenu, hNewMenu ;
hOldMenu = GetMenu(hWnd) ; // Lấy chỉ danh menu hiện tại
hNewMenu = LoadMenu(hInst , "NewMenu"); // Nạp menu mới
SetMenu(hWnd, hNewMenu); // Thay menu mới
SetMenu(hWnd, hOldMenu);  // Khôi phục menu cũ
```

§3. PHÍM CẤP TỐC

Phím lệnh cấp tốc (hay còn gọi là phím nóng) sau khi gắn cho một mục menu, có thể dùng để chọn mục menu tương ứng ngay cả khi không nhìn thấy trên màn hình. Để gắn các phím lệnh cấp tốc cho các mục menu, trước hết phải khai báo chúng trong tệp tài nguyên như sau:

```
TableName  ACCELERATORS
 {
   Key1, MenuID1 [, type] [ , option]
   Key2, MenuID2 [, type] [ , option]
   ...
   Keyn, MenuIDn [, type] [ , option]
 }
```

Trong đó:

+ TableName là tên bảng phím cấp tốc

+ ACCELERATORS là từ khoá

+ Các tham số Key1, Key2, ..., Keyn là tên các phím, chúng được gắn cho các mục thực đơn có định danh lần lượt là MenuID1, MenuID2, ..., MenuIDn

Cách viết tên phím như sau:

Đối với phím ký tự: Viết ký tự tương ứng trong 2 dấu nháy kép, ví dụ:

"A" - Phím A

Hoặc thêm dấu ^ để biểu thị sự kết hợp với phím Ctrl, ví dụ:

"^X" - Tổ hợp Ctrl X

Đối với phím ảo: Viết macro của phím ảo (xem bảng dưới), ví dụ:

VK_F1 - Phím F1

+ Tham số type cho biết phím cấp tốc là phím ký tự hay phím ảo. Nếu bỏ qua type thì đó là phím ký tự, nếu chọn type bằng VIRTKEY thì đó là phím ảo.

+ Tham số option chọn một trong các macro sau:

NOINVERT - Không làm sáng mục menu được chọn bằng phím cấp tốc

ALT - Kết hợp với phím ALT

SHIFT - Kết hợp với phím shift

CONTROL - Kết hợp với phím Ctrl

Bảng mã phím ảo

Macro	Phím
VK_CANCEL	Ctrl-Break
VK_BACK	Backspace
VK_TAB	Tab
VK_RETURN	Enter
VK_SHIFT	Shift
VK_CONTROL	Ctrl
VK_MENU	Alt
VK_CAPITAL	Caps lock
VK_ESCAPE	Esc
VK_SPACE	Spacebar
VK_PRIOR	Page Up
VK_NEXT	Page Down
VK_END	Enf
VK_HOME	Home
VK_LEFT	Left arrow
VK_UP	Up right
VK_RIGHT	Right arrow
VK_DOWN	Down arrow
VK_INSERT	Insert
VK_DELETE	Delete
VK_F1	F1
...	
VK_F12	F12

Ví dụ

```
MyMenu   MENU   DISCARDABLE
   {
    POPUP   "&File"
     {
```

```
        MENUITEM "&New\tCtrl+N", IDM_F_NEW
        MENUITEM "&Open\tCtrl+O", IDM_F_OPEN
        MENUITEM SEPARATOR
        MENUITEM "&Save\tF2", IDM_F_SAVE , GRAYED
        MENUITEM "Save &As\tCtrl+F2", IDM_F_SAVEAS, GRAYED
      }
      POPUP "&Edit"
        {
        MENUITEM "&Copy\tCtrl+C", IDM_E_COPY
        MENUITEM " C&ut\tAlt+C", IDM_E_CUT
        MENUITEM "&Paste\tInsert", IDM_E_PASTE, GRAYED
        }
      MENUITEM "E&xit\tEnd", IDM_EXIT
    }
    // Khai báo bảng phím cấp tốc và gắn với các mục menu
  MyMenu  ACCELERATORS
    {
     "^N", IDM_F_NEW
     "^O", IDM_F_OPEN
     VK_F2, IDM_F_SAVE , VIRTKEY
     VK_F2, IDM_F_SAVEAS, VIRTKEY, CONTROL
     "^C", IDM_E_COPY
     "C", IDM_E_CUT, ALT
     VK_INSERT, IDM_E_PASTE, VIRTKEY
     VK_END, IDM_EXIT, VIRTKEY
    }
```

Ý nghĩa như sau:

Ctrl + N	chọn nút lệnh New
Ctrl + O	chọn nút lệnh Open
F2	chọn nút lệnh Save
Ctrl + F2	chọn nút lệnh Save As
Ctrl + C	chọn nút lệnh Copy
Alt + C	chọn nút lệnh Cut
Insert	chọn nút lệnh Paste
End	chọn nút lệnh Exit

Để sử dụng được các phím cấp tốc còn cần sửa đổi hàm WinMain như sau:

+ Khai báo một biến để chứa chỉ danh bảng phím cấp tốc:

```
HACCEL  hAccel ;
```

+ Trước vòng lặp nhận thông điệp, cần nạp bảng phím cấp tốc bằng câu lệnh:

```
hAccel = LoadAccelerators(hInst, "MyMenu") ;
```

ở đây:

hInst là chỉ danh bản sao chương trình

MyMenu là tên bảng phím cấp tốc (thường trùng với tên menu)

+ Vòng lặp nhận thông điệp cần sửa như sau:

```
while (GetMessage(&msg, NULL, 0, 0) )
  {
    if(!TranslateAccelerator(hWnd, hAccel, &msg))
     {
       TranslateMessage(&msg);
       DispatchMessage(&msg);
     }
  }
```

Hàm

```
TranslateAccelerator(hWnd, hAccel, &msg)
```

có nhiệm vụ chuyển đổi phím cấp tốc thành thông điệp WM_COMMAND. Hàm cho giá trị TRUE nếu phím cấp tốc được nhấn, và giá trị FALSE nếu trái lại.

§4. XỬ LÝ SỰ KIỆN CHỌN NÚT LỆNH CỦA MENU

Mỗi khi người sử dụng chọn một nút lệnh trên menu (bằng chuột hay phím lệnh cấp tốc), hàm WinProc sẽ được gọi và được nhận các giá trị sau:

```
Message =  WM_COMMAND
LOWORD(wParam) = Định danh của nút lệnh được chọn
```

Để phân loại và xử lý các sự kiện chọn nút lệnh, ta sử dụng toán tử switch theo mẫu sau (trong hàm WndProc):

```
switch(Message)
  {
    case WM_COMMAND :  // Một nút lệnh được chọn
```

```
        switch (LOWORD(wParam)) // Rẽ nhánh theo nút lệnh
         {
          case IDM_COM_1 :
    // Đoạn chương trình ứng với nút lệnh có định danh IDM_COM_1
          break;
          case IDM_COM_2 :
    // Đoạn chương trình ứng với nút lệnh có định danh IDM_COM_2
          break;
          ...
          case IDM_COM_n :
    // Đoạn chương trình ứng với nút lệnh có định danh IDM_COM_n
          break;
          default:
          return DefWindowProc(hWnd, Message, wParam, lParam);
       }// Kết thúc switch phân loại và xử lý các thông điệp về nút lệnh
      break;
       // Xử lý các thông điệp khác
    } // Kết thúc switch phân loại và xử lý tất cả các thông điệp
```

§5. ĐIỀU KHIỂN TRẠNG THÁI CỦA MENU

Trạng thái ban đầu của các mục menu được xác định khi thiết kế menu (xem §1). Sau đó chúng ta có thể thay đổi trạng thái của các mục menu bằng các hàm sau:

5.1. Lấy chỉ danh menu

+ Khai báo biến để chứa chỉ danh menu

```
HMENU  hMenu, hSubMenu ;
```

+ Nạp menu vào bộ nhớ và xác định chỉ danh menu được nạp:

```
hMenu = LoadMenu(hInst, "MenuName");
```

trong đó:

hInst là chỉ danh bản sao chương trình

MenuName là tên menu (xem phần thiết kế menu)

+ Lấy chỉ danh menu đang gắn với cửa sổ:

```
hMenu = GetMenu(hWnd)
```

trong đó:

hWnd là chỉ danh cửa sổ

+ Lấy chỉ danh menu con:

hSubMenu = GetSubMenu(hMenu, n)

trong đó:

- hMenu là chỉ danh menu
- n là số thứ tự của menu con (tính từ 0)

5.2. Thay đổi trạng thái các mục menu, dùng hàm:

BOOL EnableMenuItem(HMENU hMenu,UINT nMenuItem,UINT Status)

trong đó:

- hMenu là chỉ danh menu
- nMenuItem cho biết mục menu cần thay đổi trạng thái, nó có thể là định danh hoặc số thứ tự (tính từ 0) của mục menu tuỳ thuộc vào giá trị của tham số nStatus. Trong tham số nStatus có chứa macro MF_BYCOMMAND thì nMenuItem là định danh, còn nếu tham số nStatus chứa macro MF_BYPOSITION thì nMenuItem là số thứ tự.
- nStatus là một trong 2 macro MF_BYCOMMAND, MF_BYPOSITION kết hợp với một trong 3 macro:

MF_ENABLED , MF_DISABLED , MF_GRAYED

Ví dụ có thể đặt nStatus bằng

MF_BYCOMMAND | MF_ENABLED

hoặc bằng

MF_BYPOSITION | MF_GRAYED

Chú ý: Khi không sử dụng cả 2 macro MF_BYCOMMAND và MF_BYPOSITION thì mặc nhiên xem như MF_BYCOMMAND được sử dụng.

Các macro MF_ENABLED, MF_DISABLED và MF_GRAYED xác định trạng thái mới của mục menu:

MF_ENABLED	- Mục menu được phép chọn
MF_DISABLED	- Mục menu không cho chọn
MF_GRAYED	- Mục menu bị mờ và không cho chọn

5.3. Đánh dấu các mục menu, dùng hàm:

DWORD CheckMenuItem(HMENU hMenu,UINT nMenuItem,UINT nCheck);

trong đó:

hMenu là chỉ danh menu

nMenuItem cho biết mục menu cần đánh dấu hay bỏ đánh dấu, nó được viết giống như trong hàm EnableMenuItem

nCheck (giống như nStatus) là sự kết hợp của một trong 2 macro

MF_BYCOMMAND, MF_BYPOSITION

và một trong 2 macro:

MF_CHECKED (đánh dấu)

MF_UNCHECKED (bỏ đánh dấu)

- Giá trị của hàm: là trạng thái cũ của mục menu (MF_CHECKED hoặc MF_UNCHECKED)

§6. THÊM BỚT CÁC MỤC TRÊN MENU

6.1. Bổ sung một mục vào cuối một menu

+ Hàm AppendMenu dùng để bổ sung một mục mới vao cuối một menu hoặc menu con. Mục mới có thể là nút lệnh, đường phân cách hoặc menu con.

- Nguyên mẫu

```
BOOL    AppendMenu(HMENU hMenu, UINT uFlags,
            UINT  uIDNewItem , LPSTR lpContentShow)
```

- Các đối:

hMenu là chỉ danh menu cần bổ sung

uFlags xác định:

* Mục thêm vào là nút lệnh, menu con hay đường phân cách
* Dạng thể hiện của mục thêm vào là chuỗi hay ảnh bitmap
* Trạng thái thể hiện là: Cho chọn, không cho chọn, mờ,...

Giá trị của uFlags là tổ hợp của các macro thuộc 3 nhóm sau:

Nhóm 1

MF_POPUP : Thêm vào menu con

MF_SEPARATOR : Thêm vào đường phân cách

Nếu bỏ trống : Thêm vào nút lệnh

Nhóm 2

MF_STRING : Thể hiện dưới dạng chuỗi

MF_BITMAP : Thể hiện dưới ảnh bitmap

Nếu bỏ trống : Thể hiện dưới dạng chuỗi

Nhóm 3 : Trạng thái thể hiện

MF_CHECKED

MF_UNCHECKED

MF_GRAYED

MF_ENABLED

MF_DISABLED

uIDNewItem là:

định danh của nút lệnh,

hoặc chỉ danh của menu con,

hoặc là số 0 nếu thêm vào đường phân cách

lpContentShow xác định nội dung hiển thị của mục (chuỗi hay ảnh bitmep) và nó có thể là:

Địa chỉ chuỗi , nếu mục thêm vào hiện thị dạng chuỗi

Chỉ danh của bitmap, nếu thể hiện bằng ảnh bitmap

Chuỗi rỗng nếu mục thêm vào là đường phân cách

+ Gía trị hàm: Nếu thành công, hàm trả về giá TRUE (khác 0)

6.2.Chèn một mục vào menu

+ Hàm InsertMenu dùng để chèn một mục mới vào một vị trí nào đó trong một menu hoặc menu con. Mục mới có thể là nút lệnh, đường phân cách hoặc menu con. Hai hàm ApendMenu và InsertMenu đều có tác dụng bổ sung một mục mới vào menu, sự khác nhau chỉ là bổ sung vào đâu. Hàm thứ nhất luôn bổ sung vào cuối, còn hàm thứ hai sẽ có thêm một đối để xác định vị trí của mục mới thêm vào.

- Nguyên mẫu

```
BOOL    InsertMenu(HMENU hMenu , UINT uPosition , UINT uFlags,
                   UINT  uIDNewItem , LPSTR lpContentShow)
```

- Các đối:

hMenu, uFlags, uIDNewItem và lpContentShow

có ý nghĩa như trong hàm AppendMenu

- Đối uPosition xác định một mục trong menu, mà trước nó sẽ chèn mục mới. Giá trị của uPosition có thể là:

định danh của mục nếu đối uFlags không chứa macro

MF_BYPOSITION

hoặc vị trí (tính từ 0) của mục nếu đối uFlags chứa macro

MF_BYPOSITION

+ Gía trị hàm: Nếu thành công, hàm trả về giá TRUE (khác 0)

6.3. Xoá một mục menu

Hàm DeleteMenu có tác dụng xoá (bỏ đi) một mục menu (nút lệnh, menu con hoặc đường phân cách)

+ Nguyên mẫu:

```
BOOL DeleteMenu(HMENU hMenu, UINT uPosition, UINT uFlags)
```

+ Các đối:

hMenu là chỉ danh menu

uPosition cho biết mục cần xoá, nó là định danh hoặc số thứ tự (tính từ 0) của mục cần xoá, tuỳ thuộc vào giá trị của đối uFlags là MF_BYCOMMAND hay MF_BYPOSITION

uFlags cần là một trong 2 macro: MF_BYCOMMAND và MF_BYPOSITION

+ Gía trị hàm: Nếu thành công, hàm trả về giá TRUE (khác 0)

6.4. Sửa đổi tên mục, chỉ danh và trạng thái

Hàm ModifyMenu cho phép đồng thời thay đổi tên, định danh và trạng thái của một mục menu

+ Nguyên mẫu:

```
BOOL ModifyMenu(HMENU hMenu, UINT uPosition, UINT uFlags,
                UINT uNewID, LPSTR lpNewContentShow) ;
```

+ Các đối:

- hMenu là chỉ danh menu
- uPosition xác định mục cần thay đổi, nó có thể là:

Định danh của nút lệnh hoặc chỉ danh của menu con, nếu uFlags chứa macro

MF_BYCOMMAND (mặc định)

Hoặc số thứ tự của mục (tính từ 0), nếu uFlags chứa macro

MF_BYPOSITION

- uNewID xác định định danh mới hay chỉ danh mới, nó có thể là:

Định danh mới của nút lệnh, nếu uFlags không chứa macro MF_POPUP

Chỉ danh mới của menu con, nếu uFlags chứa macro MF_POPUP

- lpNewContentShow xác định nội dung hiển thị mới của mục, nó có thể nhận các giá trị:

Địa chỉ của một chuỗi chứa tiêu đề mới của mục, nếu uFlags chứa macro

MF_STRING (đây là giá trị mặc định)

Địa chỉ của một xâu rỗng, nếu uFlags chứa macro

MF_SEPARATOR

Chỉ danh của một bitmap, nếu uFlags chứa macro

MF_BITMAP

- uFlags là tổ hợp của 4 nhóm sau (mỗi nhóm lấy tối đa một macro)

Nhóm 1 (dùng cho đối uPosition)

BY_COMMAND (mặc định)

BY_POSITION

Nhóm 2 (dùng cho đối uNewID)

BY_POPUP

Nhóm 3 (dùng cho đối lpNewContentShow)

MF_STRING (mặc định)

MF_BITMAP

MF_SEPARATOR

Nhóm 4 (dùng để quy định trạng thái mới)

MF_CHECKED

MF_UNCHECKED

MF_GRAYED

MF_ENABLED

MF_DISABLED

+ Gía trị hàm: Nếu thành công, hàm trả về giá TRUE (khác 0)

§7. TẠO MENU MỚI TRONG CHƯƠNG TRÌNH

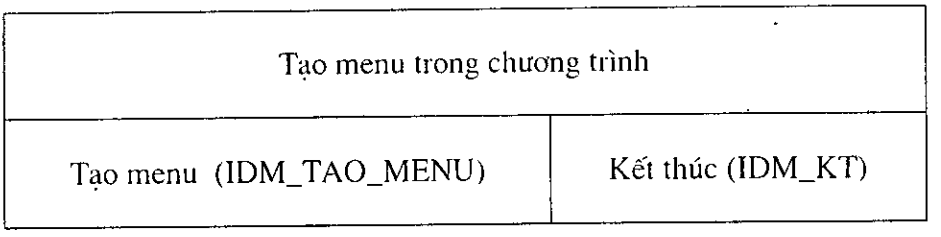
Đầu tiên dùng hàm

HMENU CreateMenu()

để tạo một menu mới, nó là menu rỗng.

Sau đó dùng các hàm AppendMenu và InsertMenu để bổ sung vào các menu con, nút lệnh, đường phân cách.

Ví dụ: Chương trình TaoMenu có giao diện ban đầu như sau:



Menu ban đầu gồm 2 nút lệnh. Nút lệnh "Kết thúc" dùng để kết thúc chương trình. Khi chọn nút lệnh "Tạo menu", sẽ nhận được menu mới như sau:

Tạo menu trong chương trình			
Chọn hình	Tính	Trở về menu cũ (IDM_MENU_CU)	Kết thúc (IDM_KT)
Tam giác (IDM_C)TG)	Diện tích (IDM_T_DT)		
Hình tròn (IDM_C_HT)	Chu vi (IDM_T_CV)		
Đoạn thẳng (IDM_C_DT)			

Trạng thái các nút lệnh: Nút "Tam giác" được chọn, nút "Chu vi" mờ

Công dụng các nút lệnh:

+ Nút "Kết thúc" dùng để kết thúc chương trình

+ Nút "Trở về menu cũ" dùng để quay trở lại menu ban đầu (với 2 nút lệnh)

+ Nút "Diện tích" dùng để đưa ra thông báo: "Tính diện tích"

+ Nút "Chu vi" dùng để đưa ra thông báo: "Tính chu vi".

+ Tác dụng của các nút lệnh của menu con "Chọn hình" như sau:

- Khi chọn nút nào, thì đánh dấu nút đó và bỏ đánh dấu 2 nút còn lại

- Nếu chọn nút "Đoạn thẳng", thì nút "Chu vi" sáng còn nút "Diện tích" mờ

- Nếu chọn 2 nút đầu, thì nút "Diện tích" sáng còn nút "Chu vi" mờ

Chương trình TaoMenu chẳng nhưng minh hoạ việc tạo một menu mới, mà còn minh hoạ cách gắn menu vào cửa sổ và cách điều khiển trạng thái menu. Chương trình gồm các tệp Resource.rc, Resource.h và TaoMenu.C như sau:

```
// Tệp Resource.rc
#include "resource.h"
#include <windows.h>
// Menu
TaoMenu MENU DISCARDABLE
BEGIN
  MENUITEM "Tao menu",  IDM_TAO_MENU
  MENUITEM "Ket thuc",   IDM_KT
END
```

// Tệp Resource.h

```
#define IDM_TAO_MENU  40001
#define IDM_KT        40002
#define IDM_C_TG      40003
#define IDM_C_HT      40004
#define IDM_C_DT      40005
#define IDM_T_DT      40006
#define IDM_T_CV      40007
#define IDM_MENU_CU   40008
```

// Tệp TaoMenu.C

```
#include <windows.h>
#include "resource.h"
HMENU hMenu, hOldMenu ;
char ten_ct[] = "taomenu";
HINSTANCE hInst;
HWND hWndMain;
LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
BOOL CALLBACK TGDlgProc(HWND,UINT,WPARAM,LPARAM);
BOOL CALLBACK HTDlgProc(HWND,UINT,WPARAM,LPARAM);
int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,
                                    LPSTR lpszCmdLine,int nCmdShow)
 {
  MSG msg;  WNDCLASS wc;
  hInst = hInstance;
   memset(&wc, 0, sizeof(WNDCLASS));
    wc.style=CS_HREDRAW|CS_VREDRAW;
    wc.lpfnWndProc = WndProc;
    wc.hInstance = hInst;
    wc.lpszClassName = ten_ct;
      // wc.lpszClassName = MAKEINTRESOURCE(100);
    wc.lpszMenuName = ten_ct;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hIcon = LoadIcon(NULL,IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL,IDC_ARROW);
```

```
    wc.hbrBackground= GetStockObject(WHITE_BRUSH);
    if(!RegisterClass(&wc))
     {
      MessageBeep(0xFFFFFFFF);
      MessageBox(NULL,"Loi DK lop",NULL,MB_OK);
      return 0;
     }
  hWndMain= CreateWindow(ten_ct,
            "Tao menu trong chuong trinh",
            WS_OVERLAPPEDWINDOW,
            CW_USEDEFAULT, CW_USEDEFAULT,
            CW_USEDEFAULT, CW_USEDEFAULT,
            HWND_DESKTOP, NULL,hInst,NULL);
  if(hWndMain==NULL)
   {
     MessageBox(NULL,"Loi tao CS",NULL,MB_OK);
     return 0;
   }
  ShowWindow(hWndMain,nCmdShow);
  UpdateWindow(hWndMain);
  while(GetMessage(&msg,NULL,0,0))
   {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
   }
  UnregisterClass(ten_ct,hInst);
  return msg.wParam;
 }
 LRESULT CALLBACK WndProc(HWND hWnd,UINT Message,
WPARAM wParam, LPARAM lParam)
  {
  switch(Message)
   {
    case WM_COMMAND:
     switch(wParam)
```

```
        {
         case IDM_TAO_MENU:
         {
          // MessageBox(hWnd,"Tao menu", NULL, MB_OK);
          HMENU hMenu1, hMenu2 ;
          hOldMenu = GetMenu(hWnd);
          hMenu = CreateMenu();
          hMenu1 = CreateMenu();
          hMenu2 = CreateMenu();
          AppendMenu(hMenu,MF_STRING,IDM_KT,"Ket thuc");
          InsertMenu(hMenu, IDM_KT, MF_STRING, IDM_MENU_CU,
"Tro ve menu cu");
          InsertMenu(hMenu,IDM_MENU_CU,MF_POPUP,(UINT)hMen
u2, "Tinh");
          InsertMenu(hMenu,(UINT)hMenu2,MF_POPUP,(UINT)hMenu1,
"Chon hinh");
         AppendMenu(hMenu1,MF_STRING|MF_CHECKED, IDM_C_TG,
"Tam giac");
         AppendMenu(hMenu1,MF_STRING,IDM_C_HT,"Hinh tron");
         AppendMenu(hMenu1,MF_SEPARATOR,0,"");
         AppendMenu(hMenu1,MF_STRING,IDM_C_DT,"Doan thang");
         AppendMenu(hMenu2,MF_STRING,IDM_T_DT,"Dien tich");
         AppendMenu(hMenu2,MF_SEPARATOR,0,"");
         AppendMenu(hMenu2, MF_STRING|MF_GRAYED, IDM_T_CV,
"Chu vi");
         SetMenu(hWnd,hMenu);
         }
         break;
       case IDM_KT:
         PostMessage(hWnd,WM_CLOSE,0,0L);
         break;
        case IDM_MENU_CU:
         SetMenu(hWnd,hOldMenu);
         break;
        case IDM_T_DT:
         MessageBox(hWnd,"Tinh dien tich",NULL,MB_OK);
```

```
            break;
          case IDM_T_CV:
            MessageBox(hWnd,"Tinh chu vi",NULL,MB_OK);
            break;
          case IDM_C_TG:
          case IDM_C_HT:
          case IDM_C_DT:
            CheckMenuItem(hMenu,IDM_C_TG,MF_UNCHECKED);
            CheckMenuItem(hMenu,IDM_C_HT,MF_UNCHECKED);
            CheckMenuItem(hMenu,IDM_C_DT,MF_UNCHECKED);
            CheckMenuItem(hMenu,wParam,MF_CHECKED);
           if(wParam==IDM_C_DT)
             {
              EnableMenuItem(hMenu,IDM_T_DT,MF_GRAYED);
              EnableMenuItem(hMenu,IDM_T_CV,MF_ENABLED);
             }
           else
             {
              EnableMenuItem(hMenu,IDM_T_CV,MF_GRAYED);
               EnableMenuItem(hMenu,IDM_T_DT,MF_ENABLED);
             }
            break;
          default:
            return DefWindowProc(hWnd,Message,wParam,lParam);
            break;
        }
       break;
      case WM_CLOSE:
       DestroyWindow(hWnd);
       PostQuitMessage(0);
       break ;
      default:
         return DefWindowProc(hWnd,Message,wParam,lParam);
   }
  return 0L;
}
```

§8. MENU CÓ HÌNH ẢNH

Có thể đưa ảnh bitmap vào một mục menu (menu con, nút lệnh, đường phân cách) bằng 2 cách:

Cách 1: Dùng hàm AppendMenu hay InsertMenu để bổ sung một mục có hình ảnh (nội dung hiển thị là một ảnh bitmap) vào một menu

Cách 2: Dùng hàm ModifyMenu để thay đổi nội dung hiển thị của một mục từ dạng chuỗi sang dạng ảnh bitmap

Sau đây là một ví dụ về cách thứ 2. Giả sử trong thư mục C:\Windows chứa tệp ảnh Circles.bmp . Để đưa ảnh này vào menu, ta làm như sau:

Trong tệp RC định nghĩa một bitmap:

```
CIR_BMP BITMAP   c:\windows\circles.bmp
```

Trong tệp C dùng các câu lệnh sau để đưa hình ảnh vào nút lệnh IDM_XOA_CS

```
HMENU hMenu;
HBITMAP hBitmap;
hMenu = GetMenu(hWnd);
hBitmap = LoadBitmap(hInst,"CIR_BMP");
ModifyMenu(hMenu,IDM_XOA_CS,MF_BITMAP, IDM_XOA_CS, hBitmap);
```

§9. TÓM TẮT MỘT SỐ HÀM VỀ MENU

1. BOOL ApendMenu(HMENU hMenu, UINT uFlags, UINT uIDNewItem, LPSTR lpContentShow);
2. DWORD CheckMenuItem(HMENU hMenu, UINT uIDCheckItem, UINT uCheck)
3. BOOL CheckMenuRatioItem(HMENU hMenu,UINT idFirst, UINT idLast, UINT idCheck, UINT uFlags);

Công dụng: Đánh dấu (kiểu Ratio) mục idCheck và bỏ đánh dấu từ mục idFirst đến mục idLast.

Đối uFlags nhận một trong 2 macro

MF_BYCOMMAND và MF_BYPOSITION

Các đối idFirst, idLast và idCheck là định danh hoặc số thứ tự của các nút lệnh, tuỳ thuộc vào uFlags là MF_BYCOMMAND hay MF_BYPOSITION

4. HMENU CreateMenu(VOID)
5. BOOL DeleteMenu(HMENU hMenu, UINT uMenuItem, UINT uFlags)

uMenuItem là định danh (hoặc chỉ danh) hay số thứ tự của mục cần xoá, tuỳ thuộc vào uFlags là MF_BYCOMMAND hay MF_BYPOSITION

6. BOOL DestroyMenu(HMENU hMenu)

7. BOOL EnableMenuItem(HMENU hMenu, UINT uIDEnableItem, uEnable)

8. HMENU GetMenu(HWND hWnd)

9. int GetMenuItemCount(HMENU hMenu)

Công dụng: Xác định số mục của menu

10. UINT GetMenuItemID(HMENU hMenu, int nPos)

Công dụng: Xác định định danh hay chỉ danh của mục có vị trí (số thứ tự) nPos

11. UINT GetMenuState(HMENU hMenu, UINT uMenuItem, UINT uFlags)

Hàm cho biết trạng thái của mục uMenuItem

Nếu mục là menu con thì: Byte thấp (của giá trị hàm) chứa trạng thái, byte cao chứa số mục của menu con

12. BOOL GetMenuItemInfo(HMENU hMenu, UINT uItem, BOOL fByPosition, LPMENUITEMINFO lpInfo)

Hàm cho biết các thông tin về mục uItem

uItem là định danh hay vị trí tuỳ thuộc fByPosition là FALSE hay TRUE

Các trường của cấu trúc kiểu MENUITEMINFO do lpInfo trỏ tới sẽ chứa các thông tin liên quan về mục uItem. Sau đây là một số trường hay dùng:

+ cbsize chứa độ lớn của cấu trúc (theo byte)

+ fType cho biết dạng hiển thị, có thể nhận các giá trị:

MFT_BITMAP hiển thị bằng ảnh bitmap

MFT_STRING hiển thị bằng chuỗi ký tự

MFT_SEPARATOR hiển thị dưới dạng đường phân cách

+ fState chứa trạng thái, có thể:

MFS_CHECKED , MFS_UNCHECKED , MFS_DISABLED, MFS_ENABLED, MFS_GRAYED,

+ hSubMenu chỉ danh của mục nếu đó là menu con, hoặc bằng 0

+ dwTypeData nội dung hiển thị: hoặc là chỉ danh bitmap hoặc địa chỉ chuỗi

+ cch là độ dài của chuỗi hiển thị (tiêu đề của mục)

13. HMENU GetSubMenu(HMENU hMenu, int nPos)

Hàm cho chỉ danh của menu con thứ nPos (tính từ 0, từ trái sang phải)

14. BOOL InsertMenu(HMENU hMenu, UINT uPosition, UINT uFlags, UINT uIDNewItem, LPSTR lpNewItem)

15. BOOL ModifyMenu(HMENU hMenu, UINT uPosition, UINT uFlags, UINT uIDNewItem, LPSTR lpNewItem)

CHƯƠNG 4

HỘP HỘI THOẠI

§1. KHÁI NIỆM HỘP HỘI THOẠI

+ Công dụng:

- Dùng để tổ chức giao diện nhập dữ liệu và hiển thị kết quả

- Thông thường mỗi chức năng tính toán cần một hộp hội thoại để tổ chức giao diện (ví dụ tính diện tích tam giác, tính giá trị đa thức, tính n!, ...)

+ Chủng loại:

- Hộp hội thoại được xem như một loại cửa sổ, mỗi hộp thoại cần có một hàm để xử lý các sự kiện, thông điệp liên quan (gọi là hàm hộp thoại)

+ Hàm hộp thoại cũng có 4 đối và có cấu trúc tương tự như hàm cửa sổ

+ Cấu trúc hộp hội thoại: Gồm các ô điều khiển, thuộc các thể loại sau

- Nhãn (LTEXT, RTEXT, CTEXT)
- Hộp soạn thảo (EDITTEXT)
- Nút lệnh (PUSHBUTTON)
- Hộp danh sách (LISTBOX)
- Hộp lựa chọn (COMBOBOX)
- Phím radio (CONTROL "Button")
- Hộp đánh dấu (CONTROL "Button" BS_AUTOCHECKBOX)
- Hình chữ nhật (CONTROL "Static" SS_BLACKFRAME)
- Biểu tượng (ICON)
- Thanh cuộn (SCROLLBAR)

+ Các loại hộp thoại:

Modal - Chiếm đoạt quyền điều khiển của ứng dụng, nghĩa là khi đang mở hộp thoại modal thì không thể chuyển sang các chức năng khác của ứng dụng. Tuy vậy vẫn có thể chuyển sang các ứng dụng khác.

System Modal - Chiếm đoạt quyền điều khiển của hệ thống, nghĩa là khi đang mở hộp thoại system modal thì không thể chuyển sang các chức năng khác của ứng dụng, cũng không thể chuyển sang các ứng dụng khác.

Modeless - Không chiếm đoạt quyền điều khiển, nghĩa là khi đang mở hộp thoại modeless thì vẫn có thể chuyển sang các chức năng khác của ứng dụng và có thể chuyển sang các ứng dụng khác.

§2. THIẾT KẾ HỘP THOẠI

Hộp thoại được thiết kế trong tệp tài nguyên RC theo mẫu sau:

```
DlgName  DIALOG [DISCARDABLE] x, y, width, height
STYLE  Kiểu_hộp_thoại
CAPTION  Tiêu_đề
FONT 8, ".VnTime"
    BEGIN
        Định nghĩa các ô điều khiển
    END
```

Trong đó:

+ DlgName là tên hộp thoại, ví dụ TGDLG. Tên hộp thoại được dùng trong hàm DialogBox để hiển thị hộp thoại ra màn hình.

+ Thuộc tính DISCARDABLE chỉ ra rằng: Hộp thoại có thể loại ra khỏi bộ nhớ khi cần thiết, để dành bộ nhớ cho các ứng dụng khác.

+ x, y là toạ độ góc trên-trái của hộp thoại (gốc toạ độ là góc trên trái cửa sổ cha)

+ width và height là chiều rộng và chiều cao của hộp thoại. Đơn vị chiều rộng là 1/4 độ rộng trung bình của con chữ hệ thống. Đơn vị chiều cao là 1/8 độ cao trung bình của con chữ hệ thống.

+ Kiểu_hộp_thoại là tổ hợp của các giá trị trong bảng dưới đây

+ Tiêu_đề là một chuỗi ký tự (đặt trong nháy kép), sẽ được hiển thị trên thanh tiêu đề của hộp thoại

+ Câu lệnh FONT xác định font chữ và cỡ chữ, trong mẫu trên thì font là .VnTime và cỡ chữ là 8.

+ Định nghĩa các ô điều khiển: Các loại ô điều khiển khác nhau có cách định nghĩa khác nhau, điều này sẽ nói trong các mục tiếp theo.

Ví dụ: Dưới đây là đoạn chương trình thiết kế hộp thoại:

```
TGDLG DIALOG DISCARDABLE  0, 0, 186, 95
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Dien tich tam giac"
FONT 8, "MS Sans Serif"
BEGIN
   LTEXT        "Canh day", -1, 27, 11, 31, 8
   LTEXT        "Chieu cao", -1, 26, 28, 33, 8
   LTEXT        "Dien tich", -1, 23, 47, 29, 8
   EDITTEXT     IDC_DAY, 81, 8, 40, 14, ES_AUTOHSCROLL
```

```
    EDITTEXT      IDC_CAO, 81, 26, 40, 14, ES_AUTOHSCROLL
    EDITTEXT      IDC_TG_DT, 80, 43, 40, 14, ES_AUTOHSCROLL |
                                                    WS_DISABLED
    PUSHBUTTON    "Tinh DT", IDC_TG_TINH, 13, 67, 50, 14
    PUSHBUTTON    "Thoat", IDC_TG_THOAT, 81, 66, 50, 14
END
```

Trong ví dụ trên thì:

Tên hộp thoại: TGDLG

Chiều rộng 186, chiều cao 95

Kiểu: Là sự kết hợp của nhiều kiểu (xem giải thích trong bảng dưới)

Tiêu đề: Dien tich tam giac

Font chữ: MS Sans Serif , cỡ chữ: 8

Các ô điều khiển gồm:

3 nhãn (LTEXT)

3 hộp soạn thảo (EDITTEXT)

2 nút lệnh (PUSHBUTTON)

Một số kiểu hộp thoại hay dùng

Kiểu	ý nghĩa
DS_MADALFRAME	Tạo hộp thoại modal
WS_BORDER	Có khung
WS_CAPTION	Có thanh tiêu đề
WS_CHILD	Cửa sổ con
WS_PUPUP	Tạo cửa sổ đẩy xuống
WS_MAXIMIZEBOX	Có hộp phóng to nhất
WS_MINIMIZEBOX	Có hộp thu nhỏ nhất
WS_SYSMENU	Có menu hệ thống
WS_TABSTOP	Có thể dùng Tab để chọn ô điều khiển
WS_VISIBLE	Nhìn thấy khi gọi hàm DialogBox

§3. ĐÓNG, MỞ HỘP THOẠI, HÀM HỘP THOẠI

3.1. Để mở (hiển thị) hộp thoại cần dùng hàm:

int DialogBox(HANDLE hInst, LPSTR lpDglName, HWND hWnd,
DLGPROC DlgProc)

Trong đó:

+ hInst là chỉ danh bản sao chương trình

+ lpDlgName trỏ tới một chuỗi chứa tên hộp thoại, ví dụ đối với hộp thoại thiết kế trong §2 thì giá trị của đối này là: "TGDLG"

+ hWnd là chỉ danh cửa sổ cha

+ DlgProc là tên hàm hộp thoại tương ứng (xem dưới đây)

Giá trị trả về:

Hàm DialogBox sẽ hiển thị một hộp thoại (gọi là D)

Các sự kiện trên hộp thoại D được xử lý trong một hàm hộp thoại tương ứng

Một hàm EnDialog (bên trong hàm hộp thoại tương ứng) sẽ đóng hộp thoại D và ra khỏi hàm DialogBox. Giá trị trả về của hàm DialogBox chính là tham số thứ 2 của hàm EnDialog (xem dưới đây)

3.2. Đóng hộp thoại. Dùng hàm:

BOOL EndDialog(HWND hDlg, nStatus)

Trong đó:

hDlg là chỉ danh hộp thoại cần đóng

nStatus là TRUE hoặc FALSE (xem giải thích)

Giải thích: Hàm EndDialog được dùng trong hàm hộp thoại để đóng và kết thúc sự làm việc với hộp thoại. Nó cũng có tác dụng thoát ra khỏi hàm DialogBox đã mở hộp thoại. Giá trị trả về của hàm DialogBox chính là tham số nStatus của hàm EndDialog. Thông thường giá trị này lấy là TRUE nếu thông điệp cuối cùng trong hàm cửa sổ được xử lý, trái lại là FALSE.

3.3. Hàm hộp thoại:

+ Mỗi hộp thoại cần một hàm DLG (hàm hộp thoại) để quản lý

+ Nhiệm vụ của hàm DLG là xử lý các nút lệnh và các sự kiện, thông điệp khác liên quan đến hộp thoại.

+ Hàm DLG có nguyên mẫu hoàn toàn tương tự với hàm cửa sổ, nó cũng có 4 đối như hàm cửa sổ và được khai báo như sau:

```
BOOL CALLBACK  NameProc(HWND hWnd, UINT  Message,
                        WPARAM  wParam, LPARAM  lParam);
```

+ Các sự kiện và thông điệp thường phải xử lý gồm:

- Sự kiện chọn một nút lệnh (giống như nút lệnh menu). Khi đó

  Message = WM_COMMAND

  wParam = Mã số của nút lệnh

- Chọn nút đóng hộp thoại, khi đó

  Message = WM_CLOSE

- Trước khi mở (hiển thị) hộp thoại:

  Message = WM_INITDIALOG

+ Cách thức làm việc của hàm DLG cũng giống như hàm cửa sổ. Hệ điều hành Windows sẽ gọi hàm và cung cấp cho hàm các giá trị (Message, wParam, lParam) có liên quan đến một sự kiện nào đó.

+ Về cấu trúc, cũng giống như hàm cửa sổ, hàm DLG cũng gồm 3 phần:

- Khai báo các biến dùng trong hàm (nếu có)
- Dùng lệnh switch theo wParam để phân loại và xử lý các nút lệnh
- Dùng switch theo Message để xử lý các thông điệp khác

+ Hàm DLG cũng sẽ được trình bầy theo các phần như trên.

+ Dưới đây là một ví dụ về hàm DLG dùng để quản lý hộp thoại TGDlg (được thiết kế trong §2). Tiếp theo là cách trình bầy hàm DLG này theo khuôn mẫu 3 phần.

Hàm TGDlgProc

```
BOOL CALLBACK TGDlgProc(HWND hDlg,UINT Message,WPARAM
wParam,LPARAM lParam)
 {
  switch(Message)
   {
    case WM_COMMAND:
     switch(wParam)
      {
       case IDC_TG_THOAT:
          EndDialog(hDlg,FALSE);
          break;
       case IDC_TG_TINH:
           {
```

```
            char tg[21]; double a,h,dt;
              GetDlgItemText(hDlg,IDC_DAY,tg,21);
              a = atof(tg);
            GetDlgItemText(hDlg,IDC_CAO,tg,21);
              h = atof(tg);
              dt = a*h/2;
              sprintf(tg,"%0.2lf",dt);
            SetDlgItemText(hDlg,IDC_TG_DT,tg);
            }
           break;
        default:
          return FALSE;
       }
      break;
   case WM_CLOSE:
     EndDialog(hDlg,FALSE);
     break;
   case WM_INITDIALOG:
     SetFocus(hDlg);
     break;
   default:
     return FALSE;
  }
 return TRUE;
}
```

Cách trình bầy hàm TGDlgProc

Phần 1. Khai báo biến

Không có

Phần 2. Xử lý các nút lệnh

```
// Nút lệnh IDC_TD_TINH
case IDC_TG_TINH:              {
        double a,h,dt; char tg[20];
        GetDlgItemText(hDlg,IDC_ DAY,tg,19);
        a=atof(tg);
        GetDlgItemText(hDlg,IDC_ CAO,tg,19);
```

```
            h=atof(tg);
            dt=a*h/2;
            sprintf(tg,"%0.2f",dt);
            SetDlgItemText(hDlg,IDC_TG_DT,tg);
           }
          break;
    // Nút lệnh IDC_TG_THOAT
     case IDC_TG_THOAT:
            EndDialog(hDlg,FALSE);
            break;
```

Phần 3. Xử lý các thông điệp khác

```
    case WM_CLOSE: // Xử lý thông điệp WM_CLOSE
       EndDialog(hDlg,FALSE);
       break;
    case WM_INITDIALOG: // Xử lý thông điệp WM_INITDIALOG
     SetFocus(hDlg);
    break;
```

§4. NHÃN - VĂN BẢN CỐ ĐỊNH

+ Công dụng: Nhãn hay văn bản cố định (static text) dùng để hướng dẫn thao tác cho người dùng

+ Định nghĩa: Một văn bản cố định được định nghĩa theo các mẫu:

```
LTEXT  vb, ID , x, y, r, c
CTEXT  vb, ID , x, y, r, c
RTEXT  vb, ID , x, y, r, c
```

Trong đó:

vb là nội dung văn bản, đó là một dãy ký tự đặt trong nháy kép

ID là định danh, vì nó không được dùng nên thường lấy bằng -1

x, y là toạ độ góc trên trái, r, c là chiều rộng, chiều cao của ô chứa vb

Cách tính r,c :

Độ rộng trung bình của một ký tự bằng 4, vì vậy chiều rộng cần lớn hơn số ký tự của vb nhân với 4

Độ cao trung bình của một ký tự bằng 8, vì vậy chiều cao cần không nhỏ hơn 16

Y nghĩa: Câu lệnh LTEXT sẽ hiện văn bản vb trong ô chữ nhật xác định bởi x, y, r và c, và văn bản sẽ được căn bên trái. Câu lệnh CTEXT sẽ đặt văn bản vào giữa, còn RTEXT sẽ đặt sang phải hộp chữ nhật.

Các ví dụ:

```
LTEXT  "Đáy ",        -1 , 10, 10, 16, 20
CTEXT  "Chiều cao", -1 , 10, 30, 36, 20
RTEXT  "Diện tích",  -1, 10,  50 36, 20
```

§5. HỘP SOẠN THẢO - EDIT TEXT

+ Công dụng: Dùng để nhập dữ liệu từ bàn phím và dùng để hiển thị kết quả tính được. Nghĩa là để tạo giao diện vào-ra

+ Định nghĩa: Một hộp soạn thảo được định nghĩa theo mẫu:

EDITTEXT ID , x, y, r, c , [Style]

Trong đó:

- ID là định danh, sẽ được dùng trong chương trình, nên thường sử dụng một từ có nghĩa như IDC_DAY để đặt cho mã số. Giá trị số của IDC_DAY sẽ được xác định (bằng #define) trong tệp .H

- x, y, r, c xác định một ô chữ nhật (xem §4) dùng để soạn thảo dữ liệu hoặc chứa kết quả đưa ra từ các câu lệnh. Ô chữ nhật này gọi là hộp soạn thảo

- Style là một giá trị nguyên xác định kiểu cách (tính chất) của hộp soạn thảo. Style là tổ hợp của các macro trong bảng sau:

Các kiểu hay dùng cho hộp soạn thảo

Kiểu	Ý nghĩa
ES_MULTILINE	Cho phép nhập nhiều dòng
ES_AUTOHSCROLL	Cho phép cuộn văn bản theo chiều ngang
ES_AUTOVSCROLL	Cho phép cuộn văn bản theo chiều đứng
ES_PASSWORD	hiển thị dấu * khi nhập dữ liệu
ES_LEFT	Dữ liệu nhập được căn trái
ES_RIGHT	Dữ liệu nhập được căn phải
ES_CENTER	Dữ liệu nhập được đặt vào giữa
ES_LOWERCASE	Đổi ra chữ thường khi nhập
ES_UPPERCASE	Đổi ra chữ hoa khi nhập
ES_READONLY	Chỉ xem, không cho phép sửa
WS_DISABLED	Không cho phép đặt forcus bàn phím vào hộp

Ví dụ:

```
EDITTEXT     IDC_DAY,  80,  8, 40, 14, ES_AUTOHSCROLL
EDITTEXT     IDC_CAO,  80, 28, 40, 14, ES_AUTOHSCROLL
EDITTEXT     IDC_TG_DT,80,48, 40, 14, ES_AUTOHSCROLL |
                                      WS_DISABLED
```

+ Các hàm dùng với hộp soạn thảo:

®. *Nhận văn bản từ hộp soạn thảo:*

UINT GetDlgItemText(HWND hDlg, int nID, LPSTR lpBuf, int nMax)

Công dụng: Sao văn bản từ hộp soạn thảo có định danh nID thuộc hộp thoại có chỉ danh hDlg vào vùng nhớ do lpBuf trỏ tới. Số ký tự tối đa cần sao là nMax, kể cả ký tự NULL (ký tự kết thúc chuỗi)

Giá trị trả về: Nếu thành công hàm trả về số ký tự của chuỗi được sao, không kể ký tự NULL. Nếu có lỗi hàm bằng 0

©. *Đưa văn bản ra hộp soạn thảo:*

BOOL SetDlgItemText(HWND hDlg, int nID, LPSTR lpBuf)

Công dụng: Sao văn bản từ vùng nhớ do lpBuf trỏ tới để đưa ra hộp soạn thảo có định danh nID thuộc hộp thoại có chỉ danh hDlg

3. Lấy chỉ danh của ô điều khiển

Mỗi ô điều khiển được xem như một cửa sổ con và có thể lấy chỉ danh nhờ hàm:

HWND GetDlgItem(HWND hDlg, int nID)

Công dụng: Hàm cho chỉ danh ô điều khiển có chỉ danh nID của hộp thoại với chỉ danh hDlg.

4. Sử dụng các hàm về cửa sổ cho ô điều khiển: Sau khi đã nhận được chỉ danh, ta có thể áp dụng các hàm về cửa sổ (xem §8 của chương 2) để điều khiển các ô điều khiển. Ví dụ:

+ Dùng hàm

SetFocus

để đưa focus đến một hộp soạn thảo

+ Dùng hàm

EnableWindow

để xác định trạng thái enable (được thâm nhập) hoặc disable (không được thâm nhập) của ô điều khiển

+ Dùng hàm

SendMessage

gửi thông điệp đến các ô điều khiển để yêu cầu một điều gì đó

5. Làm việc với hộp soạn thảo bằng thông điệp

Có thể làm việc với một điều khiển bất kỳ (như hộp soạn thảo, hộp danh sách, hộp lựa chon,...) bằng cách gửi một thông điệp đến nó, thông qua hàm:

LONG SendDlgItemMessage(HWND hDlg, int nIDDlgItem,
UINT Message, WPARAM wParam, LPARAM, lParam)

Trong đó:

hDlg là chỉ danh hộp thoại

nIDDlgItem là định danh ô điều khiển của hộp thoại hDlg

Message là thông điệp được gửi đến ô điều khiển nIDDlgItem

wParam là tham số thứ nhất được gửi kèm theo thông điệp

lParam là tham số thứ hai được gửi kèm theo thông điệp

Tác dụng của hàm:

Tuỳ thuộc vào bộ 3 giá trị được gửi (Message, wParam, lParam) mà ô điều khiển nIDDlgItem sẽ có được những tính chất khác nhau.

Giá trị hàm: Phụ thuộc vào bộ 3 giá trị được gửi

Dưới đây là một số bộ thông điệp (Message, wParam, lParam) thường dùng và ý nghĩa của chúng:

1. Message = EM_CANUNDO , wParam = 0 , lParam = 0

Giá trị của hàm cho biết có thể khôi phục (UNDO) thao tác cuối của người dùng hay không: TRUE (khác 0) là được

2. Message = EM_UNDO , wParam = 0 , lParam = 0

Công dụng: Khôi phục thao tác cuối cùng của người dùng

3. Message = EM_GETMODIFY , wParam = 0 , lParam = 0

Giá trị của hàm cho biết nội dung hộp soạn thảo có thay đổi hay không: TRUE (khác 0) là có

4. Message = EM_SETMODIFY, wParam = TRUE hay FALSE, lParam = 0

Công dụng : Thiết lập giá trị mới cho cờ thay đổi (modification flag) theo wParam. Nhận xét: Khi Message = EM_GETMODIFY , có thể nhận biết giá trị cờ này

5. Message = EM_SETREADONLY,wParam = TRUE hay FALSE, lParam = 0

Công dụng : Quy định trạng thái chỉ cho phép đọc. Nếu wParam = TRUE thì chỉ cho phép đọc, nếu wParam = FALSE thì cho phép đọc, ghi

6. Message = EM_LIMITTEXT , wParam = nMax, lParam = 0

Công dụng: Quy định số ký tự tối đa có thể đưa vào hộp soạn thảo là nMax. Nếu nMax = 0, thì số ký tự tối đa được hiểu là 0XFFFFFFFF

7. Message = EM_GETLINECOUNT , wParam = 0, lParam = 0

Giá trị của hàm cho biết số dòng văn bản trong hộp soạn thảo (ít nhất là 1)

8. Message = EM_GETLINE , wParam = nLine, lParam = lpstrBuf

Công dụng: Sao chép dòng thứ n Line (nLine = 0, 1, ...) và vùng nhớ do lpstrBuf trỏ tới

9. Message = EM_GETHANDLE , wParam =0, lParam = 0

Giá trị của hàm cho biết chỉ danh vùng nhớ chứa nội dung hộp soạn thảo

10. Message = EM_SETHANDLE, wParam =(HLOCAL)hMem, lParam = 0

Công dụng: Thiết lập một vùng nhớ với chỉ danh hMem để cho hộp soạn thảo dùng.

Giá trị của hàm: Không có giá trị trả về

11. Message = WM_GETTEXT , wParam = nMaxText,
lParam = (LONG)lpszText

Công dụng : Sao chép văn bản từ hộp soạn thảo vào vùng nhớ do lpszText trỏ tới, nMaxText là số ký tự cực đại cần sao chép, kể cả ký tự NULL

Giá trị hàm: là số ký tự được sao chép

12. Message = WM_SETTEXT,wParam = nMaxText, lParam= LONG)lpszText

Công dụng : Sao chép văn bản từ vùng nhớ do lpszText trỏ tới sang hộp soạn thảo , nMaxText là số ký tự cực đại cần sao chép, kể cả ký tự NULL

Giá trị hàm: là số ký tự được sao chép

13. Message = WM_GETTEXTLENGTH , wParam = 0, lParam = 0

Giá trị hàm: cho biết độ dài (theo ký tự) của văn bản trong hộp soạn thảo

14. Message = WM_CLEAR , wParam = 0, lParam = 0

Công dụng: Xoá khối văn bản được chọn

15. Message = WM_COPY , wParam = 0, lParam = 0

Công dụng: Sao chép khối văn bản được chọn vào Clipboard

16. Message = WM_CUT , wParam = 0, lParam = 0

Công dụng: Xoá khối văn bản được chọn, và đưa vào Clipboard

17. Message = WM_PASTE , wParam = 0, lParam = 0

Công dụng: Sao văn bản trong Clipboard ra hộp soạn thảo

18. Message = WM_UNDO , wParam = 0, lParam = 0

Công dụng: Giống như trường hợp: Message = EM_UNDO

§6. NÚT LỆNH (PUSHBUTTON)

+ Công dụng: Dùng để thực hiện một chức năng chương trình nào đó. Khi người dùng bấm chuột (nút trái) tại một nút lệnh, thì một thông điệp sau được gửi tới hàm hộp thoại (chứa nút lệnh):

Message = WM_COMMAND

wParam = nIDC (định danh của nút lệnh được bấm)

Dựa vào các giá trị trên, trong hàm hộp thoại dễ dàng nhận biết nút lệnh nào được chọn để viết đoạn chương trình thực hiện chức năng của nút lệnh cho đúng chỗ, theo mẫu sau:

```
switch(Message)
  {
    case WM_COMMAND:  // Chọn nút lệnh
      switch(wParam)
        {
          case IDC_1 :
          // Đoạn chương trình thực hiện chức năng của nút lệnh IDC_1
          break ;
          case IDC_2 :
          // Đoạn chương trình thực hiện chức năng của nút lệnh IDC_2
          break ;
          ....
          case IDC_k :
          // Đoạn chương trình thực hiện chức năng của nút lệnh IDC_k
          break ;
        }
  }
```

+ Định nghĩa: Nút lệnh được định nghĩa theo mẫu:

PUSHBUTTON td, ID , x, y, r, c

Trong đó:

- td là tiêu đề của nút lệnh (sẽ hiển thị trên hộp thoại), là một dầy ký tự đặt trong nháy kép, ví dụ: "Tính diện tích"

- ID là định danh, sẽ được dùng trong chương trình, nên thường sử dụng một từ có nghĩa như IDC_TINH_.DT để đặt cho định danh. Giá trị số của IDC_TINH_DT sẽ được xác định (bằng #define) trong tệp .H

- x, y, r, c xác định một ô chữ nhật (xem §4) dùng để chứa tiêu đề của nút lệnh.

Ví dụ:

```
PUSHBUTTON   "Tính DT", IDC_TINH_DT, 13, 66, 50, 14
PUSHBUTTON   "Thoát",   IDC_ THOAT,  81, 66, 50, 14
```

Chú ý:

Có thể lấy chỉ danh của nút lệnh, sau đó dùng các hàm về cửa sổ (xem mục §8, chương 2), như:

+ Hàm SetFocus để đặt focus tại nút lệnh

+ Hàm EnableWindow để xác định trạng thái enable hay disable cho nút lệnh

§7. SỬ DỤNG HỘP THOẠI MODAL

Hộp thoại modal có đặc tính cơ bản là: Chiếm đoạt quyền điều khiển của ứng dụng, nghĩa là khi đang mở hộp thoại modal thì không thể chuyển sang các chức năng khác của ứng dụng. Như vậy nếu trong chương trình có nhiều hộp thoại modal, thì không thể mở đồng thời chúng được, mà phải đóng hộp này thì mới mở được hộp khác.

Chương trình dưới đây sẽ minh hoạ cách sử dụng các hộp thoại modal.

+ Menu chương trình gồm các nút lệnh:

- "Tam giác" với định danh IDM_TG
- "Hình tròn" với định danh IDM_HT
- "Vẽ hình tròn" với định danh IDM_VE
- "Xoá hình tròn" với định danh IDM_XOA
- "Kết thúc" với định danh IDM_KT

Hai nút lệnh đầu tiên nằm trong menu con "Tính diện tích"

+ Chương trình gồm 2 hộp thoại:

- Hộp thoại với tên TGDLG dùng cho chức năng tính diện tam giác
- Hộp thoại với tên HTDLG dùng cho chức năng tính diện hình tròn

+ Chương trình có tên là CT4_1, gồm 3 tệp nguồn:

- resource.rc,
- resource.h,
- ct4_1.c

với nội dung như sau:

// **Tệp resource.rc**

```
#include "resource.h"
#include <windows.h>
// Menu
CT4_1 MENU DISCARDABLE
BEGIN
   POPUP "Tinh DT"
   BEGIN
      MENUITEM "Tam giac",          IDM_TG
      MENUITEM "Hinh tron",         IDM_HT
   END
   MENUITEM "Ve hinh tron",         IDM_VE
   MENUITEM "Xoa hinh tron",        IDM_XOA
  MENUITEM "Ket thuc",              IDM_KT
END

// Hộp thoại Tam giác
TGDLG DIALOG DISCARDABLE  0, 0, 186, 95
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Dien tich tam giac"
FONT 8, "MS Sans Serif"
BEGIN
   LTEXT          "Canh day",-1,27,11,31,8
   CTEXT          "Chieu cao",-1,26,28,33,8
   RTEXT          "Dien tich",-1,23,47,29,8
   EDITTEXT       IDC_DAY,81,8,40,14,ES_AUTOHSCROLL
   EDITTEXT       IDC_CAO,81,26,40,14,ES_AUTOHSCROLL
   EDITTEXT       IDC_TG_DT,80,43,40,14,ES_AUTOHSCROLL |
WS_DISABLED
   PUSHBUTTON     "Tinh DT",IDC_TG_TINH,13,67,50,14
   PUSHBUTTON     "Thoat",IDC_TG_THOAT,81,66,50,14
END

//Hộp thoại Hình  tròn
HTDLG DIALOG DISCARDABLE  0, 0, 186, 95
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Dien tich hinh tron"
FONT 8, "MS Sans Serif"
```

```
BEGIN
   LTEXT          "Ban kinh",-1,19,9,29,8
   LTEXT          "Dien tich",-1,21,26,29,8
   EDITTEXT       IDC_BK,74,8,40,14,ES_AUTOHSCROLL
   EDITTEXT       IDC_HT_DT,75,26,40,14,ES_AUTOHSCROLL |
WS_DISABLED
   PUSHBUTTON     "Tinh DT",IDC_HT_TINH,7,49,50,14
   PUSHBUTTON     "Thoat",IDC_HT_THOAT,74,49,50,14
END
```

//Tệp resource.h

```
#define IDC_DAY              1000
#define IDC_CAO              1001
#define IDC_TG_DT            1002
#define IDC_TG_TINH          1003
#define IDC_TG_THOAT         1004
#define IDC_BK               1004
#define IDC_HT_DT            1005
#define IDC_HT_TINH          1006
#define IDC_HT_THOAT         1007
#define IDM_TG               1008
#define IDM_HT               1009
#define IDM_KT               1010
#define IDM_VE               1011
#define IDM_XOA              1012
```

//Tệp ct4_1.c

```
#include "resource.h"
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
char ten_ct[] = "ct4_1" ;
HANDLE hInst;
HWND hWndMain;
BOOL ve_ht = FALSE ;
LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
```

```
BOOL CALLBACK TGDlgProc(HWND,UINT,WPARAM,LPARAM);
BOOL CALLBACK HTDlgProc(HWND ,UINT,WPARAM ,LPARAM);
int WINAPI WinMain(HANDLE hInstance,HANDLE hPrevInstance,
                                        LPSTR lpszCmdLine,int nCmdShow)
 {
  MSG msg;  WNDCLASS wc;
  hInst = hInstance;
    memset(&wc,0x00,sizeof(WNDCLASS));
     wc.style=CS_HREDRAWICS_VREDRAW;
     wc.lpfnWndProc = WndProc;
     wc.hInstance = hInst;
     wc.lpszClassName = ten_ct;
     wc.lpszMenuName = ten_ct;
     wc.cbClsExtra = 0;
     wc.cbWndExtra = 0;
     wc.hIcon = LoadIcon(NULL,IDI_APPLICATION);
     wc.hCursor = LoadCursor(NULL,IDC_ARROW);
     wc.hbrBackground= GetStockObject(WHITE_BRUSH);
     if(!RegisterClass(&wc))
       {
       MessageBeep(0xFFFFFFFF);
       MessageBox(NULL,"Loi DK lop",NULL,MB_OK);
       return 0;
       }
hWndMain= CreateWindow(ten_ct,
                  "Ve va tinh dien tich cac hinh",
                  WS_OVERLAPPEDWINDOW,
                  CW_USEDEFAULT,CW_USEDEFAULT,
                  CW_USEDEFAULT,CW_USEDEFAULT,
                  HWND_DESKTOP, NULL,hInst,NULL);
   if(hWndMain==NULL)
    {
      MessageBox(NULL,"Loi tao CS",NULL,MB_OK);
      return 0;
    }
```

```
    ShowWindow(hWndMain,nCmdShow);
        UpdateWindow(hWndMain);
    while(GetMessage(&msg,NULL,0,0))
     {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
     }
        memset(&wc,0x00,sizeof(WNDCLASS));
    UnregisterClass(ten_ct,hInst);
    return msg.wParam;
  }
LRESULT CALLBACK WndProc(HWND hWnd,UINT
Message,WPARAM wParam,LPARAM lParam)
    {
    PAINTSTRUCT  ps ;
        RECT rect;
        HDC hDC ;
    switch(Message)
     {
      case WM_COMMAND:
       switch(wParam)
        {
         case IDM_TG:
           DialogBox(hInst,"TGDLG",hWnd,TGDlgProc);
          break;
         case IDM_HT:
           DialogBox(hInst,"HTDLG",hWnd,HTDlgProc);
          break;
         case IDM_KT:
           PostMessage(hWnd,WM_CLOSE,0,0L);
          break;
         case IDM_VE:
           ve_ht = TRUE ;
           InvalidateRect(hWnd, NULL, TRUE);
```

```
            break;
          case IDM_XOA:
            ve_ht = FALSE ;
            InvalidateRect(hWnd, NULL, TRUE);
            break;
          default:
            return DefWindowProc(hWnd,Message,wParam,lParam);
        }// End of switch wParam
        break;
      case WM_CLOSE:
        DestroyWindow(hWnd);
        if(hWnd==hWndMain)
           PostQuitMessage(0);
        break ;
      case WM_PAINT:
        memset(&ps,0x00,sizeof(PAINTSTRUCT));
        hDC = BeginPaint(hWnd,&ps);
        SetBkMode(hDC,TRANSPARENT);
        if(ve_ht)
         {
          int x,y, a;
          GetClientRect(hWnd,&rect);
          x= rect.right/2;y= rect.bottom/2;
          if(x>y)
             a=4*y/5;
          else
             a=4*x/5;
          Ellipse(hDC,x-a,y-a,x+a,y+a);
         }
        EndPaint(hWnd,&ps);
        break ;
      default:
         return DefWindowProc(hWnd,Message,wParam,lParam);
       } // End of switch Message
  return 0L;
}
```

```
BOOL CALLBACK TGDlgProc(HWND hDlg,UINT Message,WPARAM
wParam,LPARAM lParam)
 {
  switch(Message)
   {
    case WM_COMMAND:
      switch(wParam)
       {
        case IDC_TG_THOAT:
          PostMessage(hDlg,WM_CLOSE,0,0);
          break;
        case IDC_TG_TINH:
         {
          char tg[21]; double a,h,dt;
          GetDlgItemText(hDlg,IDC_DAY,tg,21);
          a = atof(tg);
          GetDlgItemText(hDlg,IDC_CAO,tg,21);
          h = atof(tg);
         dt = a*h/2;
          sprintf(tg,"%0.2lf",dt);
          SetDlgItemText(hDlg,IDC_TG_DT,tg);
         }
         break;
       default:
        return FALSE;
       }
      break;
    case WM_CLOSE:
      EndDialog(hDlg,FALSE);
      break;
    case WM_INITDIALOG:
      SetFocus(hDlg);
      break;
     default:
```

```
        return FALSE;
    }
  return TRUE;
 }
BOOL CALLBACK HTDlgProc(HWND hDlg,UINT Message,WPARAM
wParam,LPARAM lParam)
 {
  switch(Message)
   {
    case WM_COMMAND:
      switch(wParam)
       {
        case IDC_HT_THOAT:
          PostMessage(hDlg,WM_CLOSE,0,0);
          break;
        case IDC_HT_TINH:
         {
           char tg[21]; double r,dt;
           GetDlgItemText(hDlg,IDC_BK,tg,21);
           r = atof(tg);
           dt = 3.14*r*r;
           sprintf(tg,"%0.2lf",dt);
           SetDlgItemText(hDlg,IDC_HT_DT,tg);
         }
          break;
        default:
          return FALSE;
       }
      break;
    case WM_CLOSE:
      EndDialog(hDlg,FALSE);
      break;
    case WM_INITDIALOG:
      SetFocus(hDlg);
```

```
        break;
      default:
        return FALSE;
    }
  return TRUE;
}
```

§8. HỘP THOẠI MODELESS

Hộp thoại Modeless có đặc tính cơ bản là: Không chiếm quyền điều khiển, nghĩa là khi đang mở hộp thoại modeless thì vẫn có thể chuyển sang các chức năng khác của ứng dụng và có thể chuyển sang các ứng dụng khác. Như vậy nếu trong chương trình có mở đồng thời nhiều hộp thoại modeless.

Để chuyển từ hộp thoại madal (trong §7) sang modeless cần một số thay đổi sau:

+ Khi thiết kế hộp thoại, tròng dòng STYLE cần đưa vào thuộc tính:

WS_VISIBLE

+ Để hiển thị hộp thoại, thay hàm DialogBox bằng hàm HWND CreateDialog (HANDLE hInst, LPSTR lpDglName, HWND hWnd, DLGPROC DlgProc)

Trong đó:

+ hInst là chỉ danh bản sao chương trình

+ lpDlgName trỏ tới một chuỗi chứa tên hộp thoại, ví dụ đối với hộp thoại thiết kế trong §2 thì tham số này là: "TGDLG"

+ hWnd là chỉ danh cửa sổ cha

+ DlgProc là tên hàm hộp thoại tương ứng

Giá trị trả về:

Nếu thành công hàm cho chỉ danh hộp thoại

Nếu có lỗi hàm cho giá trị NULL

+ Để đóng (huỷ) hộp thoại, thay hàm EndDialog bằng hàm:

BOOL DestroyWindow(hDlg)

Trong đó hDlg là chỉ danh hộp thoại cần đóng.

+ Sửa đổi vòng lặp đợi nhận thông điệp (cuối hàm WinMain)

Vì hàm cửa sổ luôn luôn nhận thông điệp, ngay cả khi hộp thoại modeless đang mở nên phải dùng hàm:

BOOL IsDialogMessage(HWND hDlg, LPMSG lpMsg)

để kiểm tra xem thông điệp chứa trong vùng nhớ do lpMsg trỏ tới có phải là của cửa sổ có chỉ danh hDlg hay không? Nếu đúng, thì thông điệp được chuyển cho hàm của hộp thoại và hàm IsDialogMessage trả về giá trị TRUE. Nếu không đúng, thì thông điệp không được chuyển cho hàm của hộp thoại và IsDialogMessage trả về giá trị FALSE

Như vậy, sau khi dùng hàm GetMessage để nhận thông điệp từ hàng đợi, cần dùng hàm IsDialogMessage để kiểm tra xem thông điệp đó có phải là của hộp thoại modeless hay không?

Nếu đúng thì gửi thông điệp cho hàm của hộp thoại xử lý

Nếu không phải, thì dùng hàm DispatchMessage để gửi thông điệp cho hàm cửa sổ xử lý

Vòng lặp đợi nhận thông điệp có thể viết theo mẫu:

```
while(GetMessage(&msg,NULL,0,0))
    {
      if(!IsDialogMessage( hDlgModeless, &msg))
     {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
     }
    }
```

Chú ý: Biến hDlgModeless (dùng để chứa chỉ danh hộp thoại modeless) cần khai báo như biến toàn bộ và được khởi gán bằng NULL như sau:

```
HWND nDlgModeless = NULL ;
```

Ví dụ muốn làm cho các hộp thoại trong chương trình CT4_1 (mục trên) thành modeless (khi đó có thể mở để làm việc đồng thời 2 hộp thoại và các chức năng vẽ, xoá hình tròn), ta cần thực hiện các thay đổi sau trên các tệp nguồn của CT4_1:

+ Tệp resource.h không thay đổi

+ Trong tệp resource.rc cần đưa thêm thuộc tính WS_VISIBLE vào các câu lệnh STYLE trong thiết kế 2 hộp thoại. Như vậy câu lệnh STYLE cần có nội dung sau:

```
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION |
                              WS_SYSMENU |WS_VISIBLE
```

+ Trong tệp ct4_1.c cần có các bổ sung và thay đổi sau:

- Trước hàm WinMain khai báo thêm 2 biến toàn bộ để chứa chỉ các hộp thoại modeless:

```
HWND hDlgTG, hDlgHT ;
```

- Vòng lặp đợi nhận thông điệp cần viết như sau

```
while(GetMessage(&msg,NULL,0,0))
 {
    if(!IsDialogMessage(hDlgTG,&msg) )
       {
        if(!IsDialogMessage(hDlgHT,&msg) )
         {
              TranslateMessage(&msg);
            DispatchMessage(&msg);
         }
       }
   }
```

- Để mở (hiển thị) các hộp thoại từ hàm cửa sổ, sau các dòng lệnh case IDM_TG và case IDM_HT cần thay hàm DialogBox bằng CreateDialog như sau:

```
case IDM_TG:
    hDlgTG = CreateDialog(hInst,"TGDLG", hWnd, TGDlgProc);
    break;
case IDM_HT:
    hDlgHT = CreateDialog(hInst,"HTDLG",hWnd,HTDlgProc);
     break;
```

- Trong các hàm hộp thoại, để đóng hộp thoại modeless đang mở, sau các dòng lệnh case WM_CLOSE, cần dùng hàm DestroyWindow thay hàm EndDialog:

```
case WM_CLOSE:
   DestroyWindow(hDlg);
break;
```

Chương trình CT4_2 dưới đây là bản sửa từ chương trình CT4_1 để làm cho các hộp thoại trở thành modeless:

// Tệp Resource.rc

```
#include "resource.h"
#include <windows.h>
//Menu
CT4_2 MENU PRELOAD
BEGIN
  POPUP "Tinh dien tich"
```

```
    BEGIN
      MENUITEM "Tam giac",          IDM_TG
      MENUITEM "Hinh tron",         IDM_HT
    END
    MENUITEM "Ve hinh tron",        IDM_VE
        MENUITEM "Xoa hinh tron",   IDM_XOA
        MENUITEM "Ket thuc",        IDM_KT
END
// Hop thoai tam giac
TGDLG DIALOG DISCARDABLE  0, 0, 186, 95
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION |
                              WS_SYSMENU | WS_VISIBLE
CAPTION "Dien tich tam giac"
FONT 8, "MS Sans Serif"
BEGIN
    LTEXT       "Canh day",-1,27,11,31,8
    CTEXT       "Chieu cao",-1,26,28,33,8
    RTEXT       "Dien tich",-1,23,47,29,8
    EDITTEXT      IDC_DAY,81,8,40,14,ES_AUTOHSCROLL
    EDITTEXT      IDC_CAO,81,26,40,14,ES_AUTOHSCROLL
    EDITTEXT      IDC_TG_DT,80,43,40,14,ES_AUTOHSCROLL |
WS_DISABLED
    PUSHBUTTON     "Tinh DT",IDC_TG_TINH,13,67,50,14
    PUSHBUTTON     "Thoat",IDC_TG_THOAT,81,66,50,14
END
//Hop thoai hinh tron
HTDLG DIALOG DISCARDABLE  0, 0, 186, 95
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION |
WS_SYSMENU | WS_VISIBLE
CAPTION "Dien tich hinh tron"
FONT 8, "MS Sans Serif"
BEGIN
    LTEXT       "Ban kinh",-1,19,9,29,8
    LTEXT       "Dien tich",-1,21,26,29,8
    EDITTEXT      IDC_BK,74,8,40,14,ES_AUTOHSCROLL
```

```
  EDITTEXT      IDC_HT_DT,75,26,40,14,ES_AUTOHSCROLL |
                                        WS_DISABLED
  PUSHBUTTON    "Tinh DT",IDC_HT_TINH,7,49,50,14
  PUSHBUTTON    "Thoat",IDC_HT_THOAT,74,49,50,14
END
```

// Tệp resource.h

```
#define IDC_DAY          1000
#define IDC_CAO          1001
#define IDC_TG_DT        1002
#define IDC_TG_TINH      1003
#define IDC_TG_THOAT     1004
#define IDC_BK           1004
#define IDC_HT_DT        1005
#define IDC_HT_TINH      1006
#define IDC_HT_THOAT     1007
#define IDM_TG           1008
#define IDM_HT           1009
#define IDM_KT           1010
#define IDM_VE           1011
#define IDM_XOA          1012
```

// Tệp ct4_2.c

```
#include "resource.h"
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
char ten_ct[] = "ct4_2" ;
HANDLE hInst;
HWND hWndMain;
BOOL ve_ht = FALSE ;
HWND  hDlgTG ;
HWND  hDlgHT ;
LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
BOOL CALLBACK TGDlgProc(HWND,UINT,WPARAM,LPARAM);
BOOL CALLBACK HTDlgProc(HWND ,UINT,WPARAM ,LPARAM);
```

```
int WINAPI WinMain(HANDLE hInstance,HANDLE hPrevInstance,
                                   LPSTR lpszCmdLine,int nCmdShow)
 {
  MSG msg;  WNDCLASS wc;
  hInst = hInstance;
    memset(&wc,0x00,sizeof(WNDCLASS));
     wc.style=CS_HREDRAW|CS_VREDRAW;
     wc.lpfnWndProc = WndProc;
     wc.hInstance = hInst;
     wc.lpszClassName = ten_ct;
     wc.lpszMenuName = ten_ct;
     wc.cbClsExtra = 0;
     wc.cbWndExtra = 0;
     wc.hIcon = LoadIcon(NULL,IDI_APPLICATION);
     wc.hCursor = LoadCursor(NULL,IDC_ARROW);
     wc.hbrBackground= GetStockObject(WHITE_BRUSH);
     if(!RegisterClass(&wc))
      {
       MessageBeep(0xFFFFFFFF);
       MessageBox(NULL,"Loi DK lop",NULL,MB_OK);
       return 0;
      }
hWndMain= CreateWindow(ten_ct,
                 "Ve va tinh dien tich cac hinh",
                 WS_OVERLAPPEDWINDOW,
                  CW_USEDEFAULT,CW_USEDEFAULT,
                 CW_USEDEFAULT,CW_USEDEFAULT,
                 HWND_DESKTOP, NULL,hInst,NULL);
  if(hWndMain==NULL)
   {
     MessageBox(NULL,"Loi tao CS",NULL,MB_OK);
     return 0;
   }
  ShowWindow(hWndMain,nCmdShow);
```

```
        UpdateWindow(hWndMain);
    while(GetMessage(&msg,NULL,0,0))
      {
        if(!IsDialogMessage(hDlgTG,&msg) )
          {
            if(!IsDialogMessage(hDlgHT,&msg) )
                {
                TranslateMessage(&msg);
                 DispatchMessage(&msg);
                }
          }
      }
    memset(&wc,0x00,sizeof(WNDCLASS));
    UnregisterClass(ten_ct,hInst);
    return msg.wParam;
  }
LRESULT CALLBACK WndProc(HWND hWnd,UINT
Message,WPARAM wParam,LPARAM lParam)
    {
    PAINTSTRUCT  ps ;
         RECT rect;
         HDC hDC ;
    switch(Message)
      {
       case WM_COMMAND:
        switch(wParam)
          {
           case IDM_TG:
             hDlgTG = CreateDialog(hInst,"TGDLG",hWnd,TGDlgProc);
            break;
           case IDM_HT:
             hDlgHT = CreateDialog(hInst,"HTDLG",hWnd,HTDlgProc);
            break;
           case IDM_KT:
```

```
        PostMessage(hWnd,WM_CLOSE,0,0L);
        break;
      case IDM_VE:
        ve_ht = TRUE ;
         InvalidateRect(hWnd, NULL, TRUE);
        break;
      case IDM_XOA:
        ve_ht = FALSE ;
        InvalidateRect(hWnd, NULL, TRUE);
        break;
      default:
        return DefWindowProc(hWnd,Message,wParam,lParam);
    }// End of switch wParam
    break;
  case WM_CLOSE:
    DestroyWindow(hWnd);
    if(hWnd == hWndMain)
      PostQuitMessage(0);
    break ;
  case WM_PAINT:
    memset(&ps,0x00,sizeof(PAINTSTRUCT));
    hDC = BeginPaint(hWnd,&ps);
    SetBkMode(hDC,TRANSPARENT);
    if(ve_ht)
     {
      int x,y, a;
      GetClientRect(hWnd,&rect);
      x= rect.right/2;y= rect.bottom/2;
      if(x>y)
        a=4*y/5;
      else
        a=4*x/5;
      Ellipse(hDC,x-a,y-a,x+a,y+a);
     }
```

```
        EndPaint(hWnd,&ps);
        break ;
      default:
        return DefWindowProc(hWnd,Message,wParam,lParam);
    } // End of switch Message
   return 0L;
  }
BOOL CALLBACK TGDlgProc(HWND hDlg,UINT Message,WPARAM
wParam,LPARAM lParam)
  {
   switch(Message)
    {
     case WM_COMMAND:
       switch(wParam)
        {
         case IDC_TG_THOAT:
           PostMessage(hDlg,WM_CLOSE,0,0);
           break;
         case IDC_TG_TINH:
           {
             char tg[21]; double a,h,dt;
             GetDlgItemText(hDlg,IDC_DAY,tg,21);
             a = atof(tg);
             GetDlgItemText(hDlg,IDC_CAO,tg,21);
             h = atof(tg);
             dt = a*h/2;
             sprintf(tg,"%0.2lf",dt);
             SetDlgItemText(hDlg,IDC_TG_DT,tg);
           }
           break;
         default:
           return FALSE;
        }
       break;
```

```
    case WM_CLOSE:
      DestroyWindow(hDlg);
      break;
    case WM_INITDIALOG:
      SetFocus(hDlg);
      break;
     default:
       return FALSE;
    }
  return TRUE;
 }
BOOL CALLBACK HTDlgProc(HWND hDlg,UINT Message,WPARAM
wParam,LPARAM lParam)
 {
  switch(Message)
    {
    case WM_COMMAND:
      switch(wParam)
       {
        case IDC_HT_THOAT:
          PostMessage(hDlg,WM_CLOSE,0,0);
          break;
        case IDC_HT_TINH:
         {
           char tg[21]; double r,dt;
           GetDlgItemText(hDlg,IDC_BK,tg,21);
           r = atof(tg);
          dt = 3.14*r*r;
          sprintf(tg,"%0.2lf",dt);
           SetDlgItemText(hDlg,IDC_HT_DT,tg);
         }
          break;
        default:
          return FALSE;
```

```
        }
        break;
    case WM_CLOSE:
        DestroyWindow(hDlg);
        break;
    case WM_INITDIALOG:
        SetFocus(hDlg);
        break;
    default:
        return FALSE;
    }
  return TRUE;
}
```

§9. HỘP DANH SÁCH - LISTBOX

+ Công dụng: Listbox (hộp danh sách) chứa một bảng danh sách các giá trị chuỗi (văn bản), và người dùng có thể dùng chuột để chọn một dòng trong bảng.

+ Định nghĩa: Listbox được định nghĩa theo mẫu:

LISTBOX ID , x, y, r, c , [Stype]

Trong đó:

- ID là định danh, sẽ được dùng trong chương trình, nên thường sử dụng một từ có nghĩa để đặt cho định danh. Giá trị cụ thể của định danh sẽ được xác định (bằng #define) trong tệp .H

- x, y, r, c xác định một ô chữ nhật (xem §4) dùng để hiển thị bảng danh sách.

- Style là một giá trị nguyên xác định kiểu cách (tính chất) của Listbox. Style là tổ hợp của các macro sau:

LBS_SORT - Các dòng văn bản được sắp xắp theo alphabe

LBS_NOTIFY - Cửa sổ cha nhận được thông điệp nhập (input message) khi người dùng bấm hoặc bấm đúp chuột

WS_VSCROLL - Các dòng văn bản được cuộn theo chiều dọc

WS_TABSTOP - Có thể chọn trong bảng danh sách bằng phím TAB

+ Các thao tác chính trên Listbox gồm:

Đưa các dòng văn bản mới vào Listbox

Xoá một số dòng trong Listbox

Lấy ra một dòng từ Listbox để sử dụng

Để thực hiện các thao tác trên, có thể dùng hàm SendDlgItemMessage (xem §5):

```
LONG SendDlgItemMessage(HWND hDlg, int nIDDlgItem,
                        UINT  Message, WPARAM wParam, LPARAM lParam)
```

với các bộ thông điệp sau:

1. Message = LB_ADDSTRING , wParam=0

   lParam = (LPSTR) lpszString

   Ý nghĩa: Bổ sung chuỗi (do lpszString trỏ tới) vào Listbox

2. Message = LB_GETCURSEL, wParam=0 , lParam = 0

   Ý nghĩa: Giá trị của hàm SendDlgItemMessage là chỉ số (tính từ 0) của dòng văn bản hiện hành (đang được chọn)

3. Message = LB_SETCURSEL, wParam=nIndex , lParam = 0

   Ý nghĩa: Quy định dòng có chỉ số nIndex là dòng văn bản hiện hành

4. Message = LB_GETTEXT, wParam=nIndex

   lParam = (LPSTR) lpszBuf

   Ý nghĩa: Sao dòng văn bản có chỉ số nIndex vào vùng nhớ do lpszBuf trỏ tới.

5. Message = LB_DELETESTRING, wParam=nIndex , lParam = 0

   Ý nghĩa: Xoá dòng văn bản có chỉ số nIndex.

6. Message = LB_INSERTSTRING, wParam=nIndex

   lParam = (LPSTR) lpszString

   Ý nghĩa: Chèn chuỗi văn bản (do lpszString trỏ tới) vào Listbox tại vị trí nIndex (nIndex là chỉ số của dòng văn bản mới thêm vào). Nếu nIndex = -1, thì bổ sung vào cuối danh sách.

Chú ý 1:

Để nhận dòng văn bản hiên hành của Listbox (có định danh IDC_List) và chứa vào mảng buf, có thể làm như sau:

```
int n; char buf[50] ;
n = Send DlgItemMessage(hDlg, IDC_List, LB_GETCURSEL,0,0);
Send DlgItemMessage(hDlg, IDC_List, B_GETTEXT,n,(LPARAM)buf);
```

Chú ý 2:

Có thể lấy chỉ danh của Listbox, sau đó dùng các hàm về cửa sổ, như:

+ hàm SetFocus để đặt focus tại Listbox

+ hàm EnableWindow để thiết lập trạng thái enable hay disable cho Listbox

Chương trình trong mục §6 chương 5 sẽ minh hoạ cách dùng Listbox và Combobox

§10. HỘP LỰA CHỌN - COMBOBOX

+ Công dụng: Combobox cũng giống như Listbox, cho phép dùng chuột để chọn một dòng trong một bảng danh sách. Sự khác nhau chỉ là: Bảng danh sách của Listbox luôn hiển thị, còn bảng danh sách của Combobox chỉ xuất hiện khi kích chuột vào biểu tượng mũi tên xuống ở bên phải.

+ Định nghĩa: Một Combobox được định nghĩa theo mẫu:

COMBOBOX ID , x, y, r, c , [Style]

Trong đó:

- ID là định danh, sẽ được dùng trong chương trình, nên thường sử dụng một từ có nghĩa để đặt cho định danh. Giá trị cụ thể của định danh sẽ được xác định (bằng #define) trong tệp .H

- x, y, r, c xác định một ô chữ nhật (xem §4) dùng để hiển thị bảng danh sách

- Style là một giá trị nguyên xác định kiểu cách (tính chất) của Combobox. Style là tổ hợp của các macro sau:

CBS_SORT - Các dòng văn bản được sắp xắp theo alphabe

CBS_DROPDOWN (mặc định) - Danh sách các dòng văn bản chỉ xuất hiện khi kích chuột vào biểu tượng mũi xuống ở bên phải của hộp

CBS_SIMPLE - Danh sách các dòng văn bản luôn xuất hiện (như Listbox)

WS_VSCROLL - Các dòng văn bản được cuộn theo chiều dọc

WS_TABSTOP - Có thể chọn trong bảng danh sách bằng phím TAB

+ Các thao tác chính trên Combobox gồm:

Đưa các dòng văn bản mới vào Combobox

Xoá một số dòng trong Combobox

Lấy ra một dòng để sử dụng

Để thực hiện các thao tác trên, có thể dùng hàm SendDlgItemMessage (xem §5):

LONG SendDlgItemMessage(HWND hDlg, int nIDDlgItem,
UINT Message, WPARAM wParam, LPARAM, lParam)

với các bộ thông điệp sau:

1. Message = CB_ADDSTRING , wParam=0

   lParam = (LPSTR) lpszString

   Ý nghĩa: Bổ sung chuỗi (do lpszString trỏ tới) vào Combobox

2. Message = CB_GETCURSEL, wParam=0 , lParam = 0

   Ý nghĩa: Giá trị của hàm SendDlgItemMessage là chỉ số (tính từ 0) của dòng văn bản hiện hành (đang được chọn)

3. Message = CB_SETCURSEL, wParam=nIndex , lParam = 0

   Ý nghĩa: Quy định dòng có chỉ số nIndex là dòng văn bản hiện hành

4. Message = CB_DELETESTRING, wParam=nIndex , lParam = 0

   Ý nghĩa: Xoá dòng văn bản có chỉ số nIndex.

5. Message = CB_INSERTSTRING, wParam=nIndex , lParam = (LPSTR) lpszString

Ý nghĩa: Chèn chuỗi văn bản (do lpszString trỏ tới) vào Combobox tại vị trí nIndex (nIndex là chỉ số của dòng văn bản mới thêm vào). Nếu nIndex = -1, thì bổ sung vào cuối danh sách.

Chú ý 1:

Có thể dùng hàm GetDlgItemText (xem §5) để nhận dòng văn bản được chọn của Combobox:

UINT GetDlgItemText(HWND hDlg, int nID, LPSTR lpBuf, int nMax)

Chú ý 2:

Có thể lấy chỉ danh của Combobox, sau đó dung các hàm về cửa sổ, như:

+ Hàm SetFocus để đặt focus tại Combobox

+ Ham EnableWindow để thiết lập trạng thái enable hay disable cho Combobox

§11. CHECKBOX, PHÍM RADIO, NHÓM CÁC PHÍM RADIO

1. Hộp đánh dấu (Checkbox)

+ Công dụng: Checkbox dùng để đánh dấu hoặc bỏ đánh dấu (bằng cách bấm chuột)

+ Định nghĩa: Checkbox được định nghĩa theo mẫu:

CONTROL td, ID, "Button", BS_AUTOCHECK | WS_TABSTOP, x, y,r, c

Trong đó:

- td là tiêu đề, đó là một dãy ký tự đặt trong nháy kép

- ID là định danh, sẽ được dùng trong chương trình, nên thường sử dụng một từ có nghĩa để đặt cho định danh. Giá trị cụ thể của định danh sẽ được xác định (bằng #define) trong tệp .H

- x, y, r, c xác định một ô chữ nhật (xem $4) dùng để hiển thị tiêu đề.

2. Phím radio (Radio Button)

+ Công dụng: Phím radio dùng để nhấn xuống hoặc nhấn lên (bằng cách bấm chuột)

+ Định nghĩa: Một phím radio được định nghĩa theo mẫu:

CONTROL td, ID , "Button", BS_AUTORADIOBUTTON, x, y, r, c

Trong đó:

- td là tiêu đề của phím, đó là một dãy ký tự đặt trong nháy kép

- ID là định danh, sẽ được dùng trong chương trình, nên thường sử dụng một từ có nghĩa để đặt cho định danh. Giá trị cụ thể của định danh sẽ được xác định (bằng #define) trong tệp .H

- x, y, r, c xác định một ô chữ nhật (xem $4) dùng để hiển thị tiêu đề.

3. Nhóm các phím radio

+ Công dụng: Dùng để tạo một nhóm các phím radio loại trừ nhau (tại mỗi thời điểm chỉ có một phím ở trạng thái bị nhấn)

+ Định nghĩa: Một nhóm các phím radio được định nghĩa theo mẫu:

GROUPBOX td, ID , x, y, r, c

Trong đó:

- td là tiêu đề của nhóm, đó là một dãy ký tự đặt trong nháy kép

- ID là định danh, không dùng trong chương trình, có thể cho bằng -1

- x, y, r, c xác định một ô chữ nhật dùng để chứa các phím radio của nhóm

4. Các hàm dùng để quản lý các hộp đánh dấu, phím radio

+ Thiết lập trạng thái của Checkbox và phím radio:

CheckDlgButton(hDlg, ID, State)

Trong đó:

hDlg là chỉ danh hộp thoại

ID là định danh của Checkbox hay phím radio

State = TRUE là đánh dấu hoặc nhấn

= FALSE là không đánh dấu hoặc không nhấn

+ Nhận trạng thái của Checkbox và phím radio:

IsDlgButtonChecked (hDlg, ID)

Trong đó:

hDlg là chỉ danh hộp thoại

ID là định danh của Checkbox hay phím radio

Giá trị hàm=TRUE có nghĩa là trạng thái đánh dấu hoặc nhấn

=FALSE có nghĩa là trạng thái không đánh dấu hoặc không nhấn

+ Có thể dùng hàm SendDlgItemMessage để thiết lập hoặc nhận biết trạng thái như sau

SendDlgItemMesage(hDlg,ID,Message,wParam,lParam)

Để thiết lập trạng thái, cho:

Message = BM_SETCHECK

wParam = TRUE : Đánh dấu hoặc nhấn

= FALSE : Không đánh dấu hoặc không nhấn

lParam = 0

Để nhận trạng thái, cho:

Message = BM_GETCHECK

wParam = 0

lParam = 0

Giá trị hàm = TRUE có nghĩa là trạng thái đánh dấu hoặc nhấn

= FALSE có nghĩa là trạng thái không đánh dấu hoặc không nhấn

§12. THANH CUỘN - SCROLLBAR

12.1. Công dụng: Cung cấp một hình thức nhập dữ liệu bằng cách kích chuột hoặc di chuyển hộp trượt

12.2. Định nghĩa: Các thanh cuộn được định nghĩa theo mẫu:

SCROLLBAR ID , x, y, r, c (thanh cuộn ngang)

SCROLLBAR ID , x, y, r, c , SBS_VERT (thanh cuộn dọc)

Trong đó:

- ID là định danh, sẽ được dùng trong chương trình, nên thường sử dụng một từ có nghĩa để đặt cho nó. Giá trị cụ thể của định danh sẽ được xác định (bằng #define) trong tệp .H

- x, y, r, c xác định một ô chữ nhật dùng làm kích thước thanh cuộn.

12.3. Các hàm dùng để quản lý thanh cuộn

1. Hàm cho giá trị nguyên đang chứa trong ô điều khiển

UINT GetDlgItemInt(HWND hDlg, int nIDDlgItem, BOOL
*lpTranslate, BOOL bSigned)

Công dụng: chuyển đổi một giá trị văn bản chứa trong ô điều khiển có định danh nIDDlgItem sang dạng nguyên. Con trỏ lpTranslate trỏ tới một biến chứa kết quả cho biết thành công hay không. Giá trị bSigned bằng TRUE thì số được xem có dấu, bằng FALSE thì số được xem không dấu. Giá trị của hàm chính là kết quả chuyển đổi.

2. Hàm đưa một giá trị nguyên ra ô điều khiển

BOOL SetDlgItemInt(HWND hDlg, int nIDDlgItem, UINT uValue,
BOOL bSigned)

Công dụng: Chuyển đổi một giá trị nguyên chứa trong biến uValue sang dạng văn bản và đưa ra ô điều khiển có định danh nIDDlgItem. Giá trị bSigned bằng TRUE thì số được xem có dấu, bằng FALSE thì số được xem không dấu. Hàm trả về TRUE hoặc FALSE nếu thành công hoặc có lỗi.

3. Hàm xác định miền gía trị của thanh cuộn

SetScrollRange(HWND hWnd, int nBar, int nMinPos, int nMaxPas,
BOOL bRedraw)

trong đó:

hWnd là chỉ danh thanh cuộn hoặc chỉ danh cửa sổ có chứa các thanh cuộn chuẩn (tuỳ thuộc vào nBar)

nBar có thể nhận:

SB_CTL - Thanh cuộn là ô điều khiển, hWnd là chỉ danh thanh cuộn

SB_HORZ : Thanh cuộn ngang của cửa sổ, hWnd là chỉ danh cửa sổ

SB_VERT : Thanh cuộn dọc của cửa sổ, hWnd là chỉ danh cửa sổ

nMinPos : Vị trí (giá trị) cực tiểu

nMaxPos : Vị trí (giá trị) cực đại

bRedraw : Cho biết có cần vẽ lại để tương thích với các thay đổi tác động trên thanh cuộn, bằng TRUE - vẽ lại, bằng FALSE - không vẽ lại

4. Hàm thiết lập vị trí của hộp trượt trên thanh cuộn

SetScrollPos(HWND hWnd, int nBar, int nPos, BOOL bRedraw)

Trong đó:

- Các tham số hWnd và nBar xác định thanh cuộn (xem hàm SetScrollRange)

- nPos là vị trí (giá trị mới) của hộp trượt

- bRedraw là cờ vẽ lại, bằng TRUE là vẽ lại cho khớp với vị trí mới của hộp trượt, bằng FALSE là không vẽ lại

5. Hàm nhận vị trí của hộp trượt trên thanh cuộn

```
int GetScrollPos(HWND hWnd, int nBar)
```

Trong đó:

- Các tham số hWnd và nBar xác định thanh cuộn (xem hàm SetScrollRange)

- Giá trị hàm cho biết vị trí hiện tại của hộp trượt

6. Các thông điệp phát ra khi tác động chuột lên thanh cuộn:

Khi dùng chuột tác động lên thanh cuộn (Bấm chuột để di chuyển hoặc kéo hộp trượt) thì thông điệp sau sẽ được gửi tới hàm cửa sổ:

Message = WM_HSCROLL (đối với thanh cuộn ngang)

hoặc

Message = WM_VSCROLL (đối với thanh cuộn dọc)

Các tham số cho thêm các thông tin sau:

+ LOWORD(wParam) Cho biết chi tiết về sự tác động, nó có thể bằng:

  SB_LINELEFT - Chuyển hộp trượt sang trái 1 đơn vị

  SB_LINERIGHT - Chuyển hộp trượt sang phải 1 đơn vị

  SB_THUMBTRACK - Chuyển hộp trượt đến vị trí mới

+ HIWORD(wParam) cho biết vị trí mới của hộp trượt

+ lParam bằng chỉ danh của thanh cuộn bị tác động

CHƯƠNG 5

XỬ LÝ CÁC THÔNG ĐIỆP

§1. THÔNG ĐIỆP MENU

Khi bấm trái chuột tại nút lệnh của menu, thì Windows sẽ gửi tới hàm cửa sổ một thông điệp với các nội dung như sau:

Message = WM_COMMAMD

wParam = Định danh của nút lệnh

Hàm cửa sổ sẽ dựa vào các giá trị này để biết nút lệnh nào được chọn và cần xử lý ra sao.

§2. THÔNG ĐIỆP LIÊN QUAN ĐẾN TRẠNG THÁI CỬA SỔ

Mỗi khi thay đổi trạng thái cửa sổ, thì Windows lại gửi tới hàm cửa sổ một thông điệp. Dựa vào nội dung các thông điệp này mà hàm cửa sổ biết được điều gì xẩy ra đối với cửa sổ và cần ứng xử ra sao. Các thông điệp loại này khá phong phú, dưới dây chỉ trình bầy một số trường hợp hay dùng.

1. Ngay sau khi hiển thị cửa sổ, một thông điệp với nội dung sau được gửi tới hàm cửa sổ:

Message = WM_CREATE

2. Ngay sau khi hiển thị cửa sổ và mỗi khi thay đổi kích thước cửa sổ, thì một thông điệp với nội dung sau được gửi tới hàm cửa sổ:

Message = WM_PAINT

3. Mỗi khi thay đổi kích thước cửa sổ, thì một thông điệp với nội dung sau được gửi tới hàm cửa sổ:

Message = WM_SIZE

LOWORD(lParam) = Chiều rộng vùng làm việc của cửa sổ

HIWORD(lParam) = Chiều cao vùng làm việc của cửa sổ

4. Khi bấm trái chuột vào nút đóng cửa sổ, thì một thông điệp với nội dung sau được gửi tới hàm cửa sổ:

Message = WM_CLOSE

5. Mỗi khi di chuyển cửa sổ, thì một thông điệp với nội dung sau được gửi tới hàm cửa sổ:

Message = WM_MOVE

LOWORD(lParam) = xPos

HIWORD(lParam) = yPos

(xPos, yPos) là toạ độ góc trên-trái của cửa sổ (so với màn hình)

§3. THÔNG ĐIỆP BÀN PHÍM

1. Khi nhấn một phím (hoặc một tổ hợp phím) trên vùng làm việc cửa sổ, thì một thông điệp với các nội dung sau được gửi tới hàm cửa sổ:

Message = WM_KEYDOWN

wParam = Mã phím ảo của phím được nhấn (xem bảng bên dưới)

LOWORD(lParam) = Số lần nhấn phím liên tục

Chú ý: Để kiểm tra xem khi đó có đồng thời ấn các phím khác (như Shift, Ctrl, Alt,...) hay không, cần dùng hàm:

int GetKeyState(int nVirtKey)

Trong đó:

nVirtKey là mã phím ảo của phím cần kiểm tra

Hàm cho giá trị âm nếu phím (xác định bởi nVirtKey) bị nhấn

2. Khi nhấn một phím ký tự (có mã ASCII) trên vùng làm việc cửa sổ, thì một thông điệp với các nội dung sau được gửi tới hàm cửa sổ:

Message = WM_CHAR

wParam = Mã ASCII của phím được bấm

LOWORD(lParam) = Số lần nhấn phím liên tục

Chú ý: Khi bấm một phím ASCII sẽ nảy sinh 2 thông điệp theo thứ tự: WM_CHAR và WM_KEYDOWN

Bảng mã phím ảo - nVirtKey

Mã	Phím
VK_CANCEL	Ctrl-Break
VK_BACK	Backspace
VK_TAB	Tab
VK_RETURN	Enter
VK_SHIFT	Shift
VK_CONTROL	Ctrl

VK_MENU	Alt
VK_CAPITAL	Caps lock
VK_ESCAPE	Esc
VK_SPACE	Spacebar
VK_PRIOR	Page Up
VK_NEXT	Page Down
VK_END	End
VK_HOME	Home
VK_LEFT	Left arrow
VK_UP	Up right
VK_RIGHT	Right arrow
VK_DOWN	Down arrow
VK_INSERT	Insert
VK_DELETE	Delete
VK_F1	F1
...	
VK_F12	F12

Chú ý:

+ Đối với các phím chữ số, thì mã phím ảo là mã ASCII

+ Đối với các chữ cái, không phân biệt hoa hay thường, mã phím ảo là mã ASCII của chữ hoa, ví dụ mã phím ảo của các chữ a và A đều là 65

Cách bắt phím. Để bắt phím có thể sử dụng các mẫu sau:

```
case  WM_KEYDOWN:
 {
   int   StatePhimA, PhimB = 66 ;
   StatePhimA = GetKeyState(65) ;
   if (wParam == PhimB && StatePhimA <0)
      {
        // Phím B được nhấn trong khi phím A đang bị đè xuống
      }
   }
case  WM_KEYDOWN:
```

```
{
  int  StateCtrl, StateAlt ;
  int  PhimPageDown = VK_NEXT ;
  StateCtrl = GetKeyState(VK_CONTROL) ;
  StateAlt = GetKeyState(VK_MENU) ;

  if (wParam == PhimPageDown && StateCtrl < 0 && StateAlt < 0)
    {
     // Phím PageDown được nhấn
     // trong khi các phím Ctrl và Alt đang bị đè xuống
    }
}
```

§4. THÔNG ĐIỆP VỀ CHUỘT

Khi thực hiện một thao tác chuột trên vùng làm việc cửa sổ, thì Windows sẽ gửi tới hàm cửa sổ 3 giá trị: Message, wParam và lParam. Giá trị Message cho biết cách thao tác chuột (như di chuyển chuột, nhấn nút trái, ...). Giá trị wParam cho biết khi xảy ra sự kiện thao tác chuột, thì phím nào và nút chuột nào đang bị nhấn. Giá trị lParam cho biết vị trị xảy ra sự kiện chuột. Cụ thể như sau:

1. Message có thể nhận các giá trị:

WM_MOSEMOVE - Di chuyển chuột
WM_LBUTTONDOWN - Nhấn nút trái
WM_LBUTTONUP - Thả nút trái
WM_LBUTTONDBKCLK - Nhấn đúp nút trái
WM_MBUTTONDOWN - Nhấn nút giữa
WM_MBUTTONUP - Thả nút giữa
WM_MBUTTONDBKCLK - Nhấn đúp nút giữa
WM_RBUTTONDOWN - Nhấn nút phải
WM_RBUTTONUP - Thả nút phải
WM_RBUTTONDBKCLK - Nhấn đúp nút phải

2. wParam cho biết các phím nào và các nút chuột nào đang bị nhấn khi xảy ra sự kiện chuột, nó có thể là tổ hợp các macro sau:

MK_CONTROL - Phím Ctrl đạng bị nhấn

MK_SHIFT - Phím Shift đang bị nhấn

MK_LBUTTON - Nút trái chuột đang bị nhấn

MK_MBUTTON - Nút giữa chuột đang bị nhấn

MK_RBUTTON - Nút phải chuột đang bị nhấn

3. lParam cho biết vị trí con trỏ chuột khi phát sinh sự kiện:

LOWORD(lParam) = xPos toạ độ ngang

HIWORD(lParam) = yPos toạ độ dọc

Chú ý 1: Chỉ có các cửa sổ có thuộc tính CS_DBLCLKS mới có thể nhận được các thông điệp liên quan đến sự kiện bấm đúp chuột. Vì vậy muốn xử lý các sự kiện này thì khi đăng ký lớp cửa sổ, cần đưa vào câu lệnh:

```
WNDCLASS wc ;
wc.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS ;
```

Chú ý 2: Để kiểm tra đồng thời các thao tác chuột và bàn phím có thể dùng mẫu:

```
case LBUTTONDOWN:
  if (wParam & MK_SHIFT && wParam & MK_CONTROL)
    {
     // Khi nhấn nút trái chuột, thì các phím Shift và Ctrl đang đồng thời bị đè xuống
    }
```

Chú ý 3: Để kiểm tra xem đã cài đặt chuột hay chưa có thể dùng mẫu:

```
  if (GetSystemMetrics(SM_MOUSEPRESENT) )
     {
       // Đã cài đặt
     }
  else
     {
      // chưa cài đặt
     }
```

§5. THÔNG ĐIỆP THỜI GIAN - WM_TIMER

5.1. Tạo đồng hồ

Có thể tạo ra các đồng hồ để ngắt chương trình sau mỗi chu kỳ xác định. Mỗi khi đồng hồ đếm về 0 thì thông điệp WM_TIMER được phát sinh. Để tạo đồng hồ dùng hàm:

UINT SetTimer(HWND hWnd, UINT nID, UINT nLength,
TIMERPROC lpTFunc)

Trong đó:

hWnd là chỉ danh cửa sổ sử dụng đồng hồ

nID là định danh đồng hồ

nLength là chu kỳ thời gian (tính bằng ms) phát sinh thông điệp WM_TIMER

lpTFunc là con trỏ đến hàm được gọi khi phát sinh thông điệp WM_TIMER.

Hàm này gọi là hàm đồng hồ, có cấu trúc tương tự như hàm cửa sổ. Nếu tham số này bằng NULL (đây là phương án hay dùng) thì sẽ không có hàm nào được gọi.

Chú ý:

+ Nếu hWnd là chỉ danh cửa sổ, thì thông điệp do đồng hồ phát sinh sẽ gửi cho hàm cửa sổ.

+ Nếu hWnd là chỉ danh hộp thoại, thì thông điệp do đồng hồ phát sinh sẽ gửi cho hàm hộp thoại.

5.2. Cách thức hoạt động của đồng hồ: Cứ sau mỗi chu kỳ thời gian, đồng hồ sẽ phát sinh một thông điệp và gửi tới hàm cửa sổ (hoặc hàm hộp thoại). Nội dung thông điệp như sau:

Message = WM_TIMER

wParam chứa định danh (nID) của đồng hồ phát sinh thông điệp

lParam chứa địa chỉ của hàm đồng hồ (nếu sử dụng)

5.3. Huỷ đồng hồ

Sau khi được tạo, đồng hồ sẽ liên tục phát thông điệp WM_TIMER cho tới khi nó bị huỷ bằng hàm:

BOOL KillTimer(HWND hWnd, UINT nID)

Trong đó:

+ hWnd là chỉ danh cửa sổ sử dụng đồng hồ

+ nID là định danh đồng hồ cần huỷ

5.4. Ứng dụng thông điệp WM_TIMER

Có thể sử dụng thông điệp WM_TIMER trong các vấn đề sau:

+ Báo thức - Sau một chu kỳ thời gian cho reo chuông hoặc phát ra các âm thanh, bản nhạc để thông báo cho người sử dụng biết về thời gian làm việc của chương trình.

+ Trong trò chơi - Điều khiển tốc độ di chuyển của một đối thủ

+ Quản lý trạng thái của các ô điều khiển trong hộp thoại

§ 6. CHƯƠNG TRÌNH SỬ DỤNG THÔNG ĐIỆP THỜI GIAN

Chương trình dưới đây minh hoạ các vấn đề sau:

+ Chương trình không có cửa sổ chính

+ Cách dùng thông điệp WM_TIMER để điều khiển trạng thái nút lệnh (lúc thì cho mờ, lúc thì cho sáng)

+ Đọc dữ liệu từ tệp để tạo ComboBox và ListBox

+ Dùng ListBox để chứa kết quả nhập dữ liệu (chứa danh sách giáo viên)

Chương trình trong thư mục: **D:\WVC6\C-Win\Combo-Timer**

//Tệp Resource.rc

```
#include "resource.h"
#include "windows.h"
GVDLG DIALOG DISCARDABLE  0, 0, 301, 186
STYLE DS_MODALFRAME | WS_MINIMIZEBOX | WS_MAXIMIZEBOX |
WS_POPUP |  WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Nhap giao vien"
FONT 8, "MS Sans Serif"
BEGIN
   LTEXT        "Ho ten",-1,7,17,31,8
   EDITTEXT  IDC_HT,42,14,76,14,ES_AUTOHSCROLL| WS_TABSTOP
   LTEXT        "Hoc vi",-1,7,39,31,8
   COMBOBOX       IDC_HV,42,35,74,50,CBS_DROPDOWN |
                  CBS_AUTOHSCROLL | WS_VSCROLL | WS_TABSTOP
   LTEXT        "Chuyen nganh",-1,3,81,47,8
   LISTBOX       IDC_CN,44,89,74,58,WS_VSCROLL | WS_HSCROLL |
                 WS_TABSTOP
   LTEXT        "Danh sach GV nhap vao",-1,155,17,108,8
   PUSHBUTTON  "Bo sung vao danh sach",IDC_BS,155,90,93,14,
                    WS_DISABLED
   PUSHBUTTON     "Ket thuc",IDC_KT,174,122,50,14
   LISTBOX    IDC_KQ,131,31,151,50, WS_VSCROLL | WS_HSCROLL
END
```

//Tệp Resource.h

```
#define IDC_HT                  1000
#define IDC_HV                  1001
#define IDC_CN                  1002
#define IDC_KQ                  1003
#define IDC_BS                  1004
#define IDC_KT                  1005
```

//Tệp ComboTimer.C

```
/*
   D:\WVC6\C-Win\Combo-Timer\ComboTimer.C
   Minh hoa:
         + Chuong trinh khong co cua so chinh
         + Doc du lieu tu tep de tao ComboBox, ListBox
         + ListBox de chua ket qua (Danh sach giao vien nhap vao)
         + Dung su kien thoi gian de dieu khien trang thai nut lenh
*/
#include "resource.h"
#include <windows.h>
#include <string.h>
#include <stdio.h>
HINSTANCE hInst;
HWND hDlgMain = NULL;
BOOL CALLBACK GVDlgProc(HWND,UINT,WPARAM,LPARAM);
Int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                        LPSTR lpszCmdLine,int nCmdShow)
 {
   MSG msg;
   hInst = hInstance;
        // Mo hop thoai
        hDlgMain= CreateDialog(hInst,"GVDLG",NULL,GVDlgProc);
   SetTimer(hDlgMain,1,1000,NULL);
   while(GetMessage(&msg,NULL,0,0))
    if(!IsDialogMessage(hDlgMain,&msg))
     {
       TranslateMessage(&msg);
       DispatchMessage(&msg);
     }
    KillTimer(hDlgMain,1);
    return msg.wParam;
 }
 BOOL CALLBACK GVDlgProc(HWND hDlg,UINT Message,WPARAM
wParam,LPARAM lParam)
 {
   HWND hBS, hHT, hKQ;
        char ht[20],hv[10],cn[12],tg[40];
        int n, i;
```

```
        FILE *fp;
        hHT=GetDlgItem(hDlg,IDC_HT);
        hKQ = GetDlgItem(hDlg,IDC_KQ);
        hBS=GetDlgItem(hDlg,IDC_BS);
  switch(Message)
    {
     case WM_COMMAND:
       switch(wParam)
         {
          case IDC_BS:
            GetDlgItemText(hDlg,IDC_HT,ht,20);
            GetDlgItemText(hDlg,IDC_HV,hv,10);
            n=SendDlgItemMessage(hDlg,IDC_CN,LB_GETCURSEL,0,0);
    SendDlgItemMessage(hDlg,IDC_CN,LB_GETTEXT,n,(LPARAM)cn);
            sprintf(tg,"%s %s %s",ht,hv,cn);
SendDlgItemMessage(hDlg,IDC_KQ,LB_ADDSTRING,0,(LPARAM)tg);
            SetDlgItemText(hDlg,IDC_HT,"");
            EnableWindow(hBS,FALSE);
            SetFocus(hHT);
            break;
          case IDC_KT:
            PostMessage(hDlg,WM_CLOSE,0,0);
            break;
          default:
            return FALSE;
         }
        break;
     case WM_CLOSE:
        DestroyWindow(hDlg);
        if(hDlg==hDlgMain)
          PostQuitMessage(0);
        break;
   case WM_TIMER:
    n=SendDlgItemMessage(hDlg,IDC_HT,WM_GETTEXTLENGTH,0,0);
        if(n>0)
           EnableWindow(hBS,TRUE);
        break;
     case WM_INITDIALOG:
```

```
        fp=fopen("Hocvi.DL","r");
        if(fp!=NULL)
          {
            fscanf(fp,"%d%*c",&n);
            for (i=1;i<=n;++i)
              {
                fgets(tg,21,fp);
SendDlgItemMessage(hDlg,IDC_HV,CB_ADDSTRING,0,(LPARAM)tg);
              }
            fclose(fp);
          }
        fp=fopen("Cmon.DL","r");
        if(fp!=NULL)
          {
            fscanf(fp,"%d%*c",&n);
            for (i=1;i<=n;++i)
              {
                fgets(tg,21,fp);
SendDlgItemMessage(hDlg,IDC_CN,LB_ADDSTRING,0,(LPARAM)tg);
              }
            fclose(fp);
          }
        EnableWindow(hKQ,FALSE);
        SendDlgItemMessage(hDlg,IDC_CN,LB_SETCURSEL,1,0);
        SendDlgItemMessage(hDlg,IDC_HV,CB_SETCURSEL,1,0);
        SetFocus(hHT);
        break;
      default:
        return FALSE;
    }
  return TRUE;
}
```

CHƯƠNG 6

HIỂN THỊ VĂN BẢN LÊN MÀN HÌNH

Chương này trình bầy cách xuất các chuỗi ký tự (dữ liệu văn bản) lên vùng làm việc của cửa sổ chính (nói cho gọn là màn hình). Trong Windows, cách thức xuất dữ liệu được thực hiện theo một lược đồ thống nhất, chung cho mọi thiết bị và mọi dạng dữ liệu (văn bản, đồ hoạ, ảnh Bitmap). Vì vậy nhiều quy tắc xuất dữ liệu văn bản lên màn hình trình bầy trong chương này, vẫn áp dụng được đối với các hình vẽ và các ảnh Bitmap.

Dưới đây khi nói dữ liệu thì hiểu theo nghĩa tổng quát, tức là đúng với cả văn bản, hình vẽ và ảnh Bitmap.

§1. CÁC BƯỚC ĐỂ XUẤT DỮ LIỆU RA MÀN HÌNH

Việc xuất dữ liệu ra màn hình (vùng làm việc cửa sổ chính) được tiến hành theo 3 bước:

+ Lấy chỉ danh ngữ cảnh thiết bị của cửa sổ đưa vào một biến kiểu HDC, ví dụ biến hDC

+ Dùng các hàm API để xuất dữ liệu ra cửa sổ thông qua hDC (tham số đầu tiên trong lời gọi các hàm API là biến hDC)

+ Giải phóng hDC khi không sử dụng nó nữa.

Trong mục này sẽ trình bầy các bước 1 và bước 3. Các hàm API để xuất dữ liệu sẽ được trình bầy trong các mục còn lại của chương 6 và chương 7.

Việc lấy và giải phóng ngữ cảnh thiết bị sẽ được tiến hành khác nhau trong 2 trường hợp: Trong các đoạn chương trình xử lý thông điệp WM_PAINT (trong WM_PAINT) và trong các đoạn chương trình xử lý các thông điệp khác WM_PAINT (ngoài WM_PAINT). Dưới đây sẽ xét từng trường hợp.

1.1. Ngoài WM_PAINT (Message <> WM_PAINT)

Để lấy chỉ danh ngữ cảnh thiết bị cửa sổ dùng hàm:

```
HDC GetDC(HWND hWnd)
```

Để giải phóng chỉ danh ngữ cảnh thiết bị cửa sổ dùng hàm:

```
int ReleaseDC(HWND hWnd, HDC hDC)
```

Ví dụ để in dòng chữ HA NOI và vẽ một hình tròn, có thể dùng các lệnh sau:

```
case IDM_XUAT:  // Xử lý nút lệnh IDM_XUAT
```

```
{
  HDC hDC ;
  hDC = GetDC(hDC);
  TextOut(hDC, 10, 10, "HA NOI", 6); // In tại vị trí (10,10)
  Ellipse(hDC, 20, 20, 300, 300); // Vẽ hình elip nội tiếp hình chữ nhật
                  // có đỉnh trên-trái (20,20) và đỉnh dưới phải (300,300)
 ReleaseDC(hWnd, hDC);
}
```

1.2. Trong WM_PAINT (Message = WM_PAINT)

Để lấy chỉ danh ngữ cảnh thiết bị cửa sổ dùng hàm:

HDC BeginPaint(HWND hWnd, LPPAINTSTRUCT lpPaint)

Để giải phóng chỉ danh ngữ cảnh thiết bị cửa sổ dùng hàm:

BOOL EndPaint(HWND hWnd, CONST PAINTSTRUCT *lpPaint)

Ví dụ để in dòng chữ HAI PHONG và vẽ một chữ nhật, có thể dùng các lệnh sau:

```
case WM_PAINT:  // Xử lý thông điệp WM_PAINT
        {
         HDC hDC ;
         PAINTSTRUCT ps;
         memset(&ps, 0, sizeof(PAINTSTRUCT));
         hDC = BeginPaint(hWnd, &ps);
         SetBkMode(hDC, TRANSPARENT);
         TextOut(hDC, 10, 10, "HAI PHONG", 9); // In tại vị trí (10,10)
         Rectangle(hDC, 20, 20, 300, 300); // Vẽ hình chữ nhật có đỉnh
                          //trên-trái (20,20) và đỉnh dưới phải (300,300)
        EndPaint(hWnd, &ps);
       }
```

§2. HÀM TEXTOUT VÀ CÁC VẤN ĐỀ LIÊN QUAN

2.1. Để hiển thị văn bản lên màn hình, dùng hàm

BOOL TextOut(HDC hDC, int x, int y, LPSTR lpStr, int n)

Trong đó:

hDC là chỉ danh ngữ cảnh thiết bị của cửa sổ

x, y là vị trí mốc (bắt đầu in)

lpStr trỏ tới chuỗi cần hiển thị

n là số ký tự cần in

Hàm trả về giá trị khác 0 (TRUE) nếu thành công, và bằng 0 (FALSE) nếu có lỗi

2.2. Các vấn đề liên quan đến kết quả làm việc của hàm TextOut gồm:

1. Cách thức căn lề văn bản (vị trí văn bản) so với điểm mốc (x, y)
2. Mầu văn bản, mầu nền của văn bản (mầu nền), chế độ mầu nền
3. Thiết lập (đặt) chiều cao, chiều rộng văn bản
4. Chọn font chữ, cỡ chữ, độ nghiêng, độ đậm, ...
5. Ngữ cảnh thiết bị ảo (DC ảo)

Các vấn đề này sẽ được xét trong các mục dưới đây.

§3. CÁCH THỨC CĂN LỀ VĂN BẢN

Để quy định vị trí văn bản trên cửa sổ so với điểm mốc, cần dùng hàm:

UINT SetTextAlign (HDC hDC, UINT AlignFlag)

trong đó

hDC là DC của thiết bị xuất (ở đây thiết bị xuất là màn hình)

AlignFlag dùng để quy định cách thức căn lề, nó là tổ hợp của 2 nhóm sau:

Nhóm căn lề theo chiều ngang gồm:

TA_LEFT (mặc định) - Căn trái, điểm mốc bên trái văn bản

TA_CENTER - Căn giữa, điểm mốc ở giữa văn bản

TA_RIGHT - Căn phải, điểm mốc bên phải văn bản

Nhóm căn lề theo chiều dọc gồm:

TA_TOP (mặc định) - Căn đỉnh, điểm mốc ở trên đỉnh văn bản

TA_BASELINE - Căn nền, điểm mốc nằm trên đường nền (gần giữa) văn bản

TA_BOTTOM - Căn đáy, điểm mốc tại đáy văn bản

Giá trị của hàm cho biết cách thức căn lề đang sử dụng

Ví dụ: Để vẽ hình tròn bán kính 100 tâm tại điểm (200, 200) và in dòng chữ HA NOI tại chính giữa hình tròn, cần dùng các câu lệnh sau:

Ngoài WM_PAINT

```
HDC hDC ;
hDC = GetDC(hDC);
Ellipse(hDC, 200 - 100, 200 - 100, 200 + 100, 200 + 100);
```

```
SetTextAlign (hDC, TA_CENTER | TA_BASELINE);
TextOut(hDC, 200, 200, "HA NOI", 6);
ReleaseDC(hWnd, hDC);
```

Trong WM_PAINT

```
HDC hDC ;
PAINTSTRUCT ps;
memset(&ps, 0, sizeof(PAINTSTRUCT));
hDC = BeginPaint(hWnd, &ps);
Ellipse(hDC, 200 - 100, 200 - 100, 200 + 100, 200 + 100);
SetTextAlign (hDC, TA_CENTER | TA_BASELINE);
TextOut(hDC, 200, 200, "HA NOI", 6);
EndPaint(hWnd, &ps);
```

§4. MẦU VĂN BẢN, MẦU NỀN, CHẾ ĐỘ MẦU NỀN

4.1. Để chọn mầu văn bản, dùng hàm

COLORREF SetTextColor(HDC hDC, COLORREF color)

trong đó

hDC là chỉ danh thiết bị của cửa sổ

color là mầu sẽ được áp dụng để in các dòng văn bản

Giá trị của hàm chính là mầu văn bản đang được sử dụng

4.2. Để chọn mầu nền (của văn bản), dùng hàm

COLORREF SetBkColor(HDC hDC, COLORREF color)

trong đó

hDC là chỉ danh thiết bị của cửa sổ

color là mầu nền sẽ được áp dụng để in các dòng văn bản

Giá trị của hàm chính là mầu nền văn bản đang được sử dụng

4.3. Để chọn chế độ mầu nền, dùng hàm

int SetBkMode(HDC hDC, int mode)

trong đó

hDC là chỉ danh thiết bị của cửa sổ

mode dùng để chọn chế độ mầu nền mới, nó có thể nhận một trong 2 giá trị:

OPAQUE (mặc định) - Nền văn bản được tô theo mầu nền hiện hành (xác định trong hàm SetBkColor)

TRANSPARENT - Nền văn bản trong suốt (không được tô)

Giá trị của hàm chính là chế độ mầu nền văn bản đang được sử dụng

4.4. Khái niệm về mầu trong Windows

Mầu được tạo bằng cách pha trộn 3 mầu cơ bản: RED, GREEN và BLUE. Để có được một mầu, dùng hàm:

COLORREF RGB(int nRed, int nGreen, int nBLue)

trong đó

nRed là số lượng mầu RED (0 <= nRed <= 255)

nGreen là số lượng mầu GREEN (0 <= nGreen <= 255)

nRed là số lượng mầu BLUE (0 <= nBlue <= 255)

Dưới đây là một số mầu hay dùng (xem thêm mục 4.1, chương 7):

RGB(255, 0, 0) - đỏ

RGB(0, 0, 255) - xanh lam

RGB(255, 0, 255) - hồng

RGB(255, 255, 255) - trắng

RGB(0, 255, 0) - xanh lá cây

RGB(0, 255, 255) - xanh lơ

RGB(255, 255, 0) - vàng

RGB(0, 0, 0) - đen

§5. XÁC ĐỊNH KÍCH THƯỚC VĂN BẢN

Khi in nhiều chuỗi ký tự trên cùng một dòng, thì cần phải biết cách xác định độ rộng văn bản, còn khi in nhiều dòng ký tự thì cần phải biết chiều cao của văn bản. Trong mục này sẽ trình bầy các khái niệm và các hàm liên quan đến kích thước văn bản.

5.1. Đơn vị đo dùng trong các hàm API

Các hàm API xuất dữ liệu ra màn hình sử dụng một hệ toạ và một đơn vị đo chung gọi đơn vị logic. Hệ toạ độ mặc định lấy điểm trên-trái vùng làm việc cửa sổ làm điểm gốc, hoành độ x tăng theo chiều từ trái sang phải và tung độ y tăng theo chiều từ trên xuống dưới. Đơn vị đo mặc định là pixel, có nghĩa là một đơn vị logic bằng một pixel. Tuy nhiên, có thể thay đổi hệ

toạ độ (gốc toạ độ, chiều tăng của x, chiều tăng của y) và đơn vị đo nhờ sử dụng hàm:

```
int SetMapMode(HDC hDC, int nMapMode)
```

Hàm này sẽ được giải thích trong chương sau.

5.2. Để lấy thông tin chung về cỡ chữ ta làm như sau:

```
TEXTMETRIC tm; // Khai báo biến cấu trúc kiểu TEXTMETRIC
               // để chứa các thông tin về cỡ chữ
GetTextMetrics(hDC, &tm) ;
```

Kết quả: Các thông tin về cỡ chữ sẽ được chứa trong các thuộc tính của biến cấu trúc tm. Sau đây là các thuộc tính hay dùng:

tm.tmMaxCharWidth = Độ rộng cực đại của ký tự

tm.tmAveCharWidth = Độ rộng trung bình của ký tự

tm.tmHeight = Chiều cao của ký tự

tm.tmExternalLeading = Khoảng trống bên trên ký tự

5.3. Khoảng cách giữa các dòng. Để xác định khoảng cách giữa các dòng ký tự, có thể dùng công thức sau:

```
int LineSpace = tm.tmHeight + tm.tmExternalLeading
```

5.4. Độ rộng và chiều cao của chuỗi ký tự. Để xác định độ rộng và chiều cao của chuỗi ký tự do con trỏ lpszStr trỏ tới, ta làm như sau:

```
SIZE size ; // SIZE có 2 thuộc tính là x và y (kiểu long)
GetTextExtentPoint32(hDC, lpszStr, strlen(lpszStr), &size);
```

Kết quả:

size.cx = độ rộng

size.cy = chiều cao

§6. LẤY CHỈ DANH FONT CHỮ

6.1. Các font chuẩn có sẵn. Để lấy chỉ danh các font chuẩn của Windows, dùng hàm:

```
HGDIOBJ GetStockObject(int nObject)
```

trong đó đối nObject cho biết font cần lấy chỉ danh. nObject có thể nhận các giá trị:

SYSTEM_FONT	- Font hệ thống
DEFAULT_GUI_FONT	- Font mặc định
ANSI_FIXED_FONT	- Font có độ rộng cố định
ANSI_VAR_FONT	- Font có độ rộng thay đổi

Ví dụ:

```
HFONT  hFont ;
hFont = (HFONT) GetStockObject(SYSTEM_FONT) ;
```

Kết quả: Lấy chỉ danh font hệ thống và chứa vào biến hFont.

6.2. Sửa font chuẩn. Có thể lấy chỉ danh font chuẩn, sau đó sửa một số thuộc tính để nhận các font mới như sau:

```
LOGFONT  lf ; // Dùng lf để chứa các thuộc tính về font
HFONT  hFont ; // Chỉ danh của font
// Lấy chỉ danh font chuẩn, ví dụ SYSTEM_FONT
hFont = (HFONT) GetStockObject(SYSTEM_FONT) ;
// Lấy thông tin về font chữ hệ thống
GetObject(hFont, &lf) ;
// Kết quả: Các thông tin về font hệ thống chứa trong các thuộc tính của lf
// Thay đổi một số thuộc tính của lf để tạo font mới
strcpy(lf.lfFaceName, ".VnTime") ;
lf.lfHeight = 14 ; // Chiều cao 14
lf.lfWidth = 0; // Windows tự xác định độ rộng cho phù hợp với chiều cao
lf.lfItalic = 1 ;  // Nghiêng
lf.lfUnderline = 1 ; // Gạch dưới
lf.lfWeight = FW_NORMAL ; // Độ đậm
  /* Thuộc tính lf.lfWeight có thể nhận các giá trị sau:
        FW_THIN
     FW_LIGHT
     FW_ EXTRALIGHT
        FW_NORMAL
        FW_MEDIUM
        FW_BOLD
     FW_EXTRABOLD
     FW_HEAVY
   */
// Tạo font chữ mới theo các thuộc tính cuat lf
hFont = CreateFontIndirect(&lf) ;
```

6.3. Sử dụng hộp thoại Font. Cách chọn thông linh hoạt và tiện lợi hơn cả sử dụng hộp hội thoại Font. Để hiển thị hộp thoại Font và lấy được chỉ danh font chữ mà người sử dụng chọn, ta làm như sau:

```
LOGFONT  lf ;  // Để chứa thuộc tính của font được chọn trên hộp thoại FONT
CHOOSEFONT  cf ; // Để mở hộp thoại Font
HFONT  hFont ; // Chứa chỉ danh font được chọn
// Khởi gán cho các thuộc tính của cf
memset(&cf, 0, sizeof(CHOOSEFONT)) ;
cf.lStructSize = sizeof(CHOOSEFONT) ;
cf.hWndOwner =  hWnd ;
cf.lpLogFont =  &lf ;
cf.Flags = CF_SCREENFONTS | CF_EFFECTS ;
cf.rgbcolors =  RGB(0, 255, 255) ;
cf.nFontType = SCREEN_FONTTYPE  ;
// Gọi hàm ChooseFont để mở hộp thoại Font
ChooseFont(&cf) ;
//Kết quả:  Các thuộc tính của font được chọn (trên hộp thoại Font)
          //   sẽ chứa trong các thuộc tính của biến lf
// Dùng  lf để lấy chỉ danh của font được chọn
hFont = CreateFontIndirect(&lf);
```

§7. CHỌN FONT CHỮ ĐỂ IN RA MÀN HÌNH

7.1. Chọn font chữ. Sau khi đã lấy được chỉ danh font chữ (xem §6), ta dùng hàm sau để chọn font chữ :

```
HFONT  SelectObject(HDC  hDC,  HFONT  hFont) ;
```

Công dụng của hàm: Chọn font chữ hFont để in ra màn hình.

Giá trị của hàm: là chỉ danh font chữ hiện đang sử dụng

7.2. Sử dụng font chữ để in văn bản ra màn hình được thực hiện theo trình tự sau:

```
// Khai báo các biến
HDC  hDC ; // Chỉ danh ngữ cảnh thiết bị của cửa sổ
HFONT  hOldFont ; // Chỉ danh font đang sử dụng
HFONT  hFont ;  // Chỉ danh font mới
```

```
// Lấy hDC
hDC = GetDC(hWnd);
// Lấy hFont
.... (xem $6)
// Chọn font mới và lưu font đang dùng
hOldFont = SelectObject(hDC, hFont) ;
// In theo font mới
  TextOut(hDC, 10, 10, "HA NOI", 6);
   // Khôi phục font cũ
    SelectObject(hDC, hOldFont) ;
   // Huỷ font mới
    DeleteObject(hFont);
    // Giải phóng DC
    ReleaseDC(hWnd, hDC);
```

§8. SỬ DỤNG DC ẢO

Việc sử dụng ngữ cảnh thiết bị ảo (DC ảo) để xuất dữ liệu lên màn hình được thực hiện theo các bước sau:

+ Tạo một DC ảo (còn gọi DC bộ nhớ) tương thích với DC màn hình
+ Xuất dữ liệu (văn bản, đồ hoạ) ra DC ảo
+ Sao dữ liệu từ DC ảo sang DC màn hình

8.1. Tạo DC ảo. Để tạo một DC ảo tương thích với DC màn hình, ta có thể làm như sau:

```
// Khai báo
  HDC  hDC ;  // Chỉ danh DC màn hình
  HDC  hMemDC ; // Chỉ danh DC ảo - còn gọi DC bộ nhớ
  HBITMAP  hBitmap ; // Chỉ danh bitmap
  HBRUSH  hBrush ;  // Chỉ danh chổi tô
  int  maxX, maxY ;  // Kích thước màn hình
 // Xác định kích thước màn hình
  maxX  =  GetSystemMetrics(SM_CXSCREEN) ;
```

```
maxY = GetSystemMetrics(SM_CYSCREEN) ;
// Tạo DC ảo tương thích với DC màn hình
hDC = GetDC(hWnd) ;
hMemDC = CreateCompatibleDC(hDC);
// Tạo một bitmap tương thích với DC màn hình
hBitmap = CreateCompatibleBitmap(hDC, maxX, maxY);
// Gắn bitmap với DC ảo
SelectObject(hMemDC, hBitmap) ;
// Lấy chỉ danh chổi tô có sẵn mầu trắng
hBrush = GetStockObject(WHITE_BRUSH) ;
// Gắn chổi tô với DC ảo
SelectObject(hMemDC, hBrush) ;
```

8.2. Xuất dữ liệu ra DC ảo. Dùng các hàm API để in văn bản, vẽ hình, xoá DC ảo

```
// Xoá DC ảo dùng hàm
PatBlt(hMemDC, 0, 0, maxX, maxY, PATCOPY);
// In văn bản ra DC ảo
TextOut(hMemDC, 10, 10, "HA NOI", 6 ) ;
// Vẽ hình elip ra DC ảo
Ellipse(hMemDC, 20, 20, 300, 400);
```

8.3. Sao dữ liệu từ DC ảo sang DC màn hình

Dùng câu lệnh

```
BitBlt(hDC, 0, 0, maxX, maxY,
       hMemDC, 0, 0,
       SRCCOPY) ;
```

Nguyên mẫu hàm BitBlt như sau:

```
BOOL BitBlt(HDC hDest, int X, int Y, int Width, int Height,
            HDC hSource, int SourceX, int SourceY,
            DWORD dwRaster)
```

trong đó:

hDest là chỉ danh DC đích (nhận)

X, Y là vị trí trên-trái nơi bitmap hiển thị

Width là độ rộng của bitmap

Height là chiều cao của bitmap

hSource là chỉ danh DC nguồn

SourceX, SourceY là toạ độ điểm trên-trái của bitmap, thường là 0, 0

dwRaster cho biết cách hiển thị từng bit của bitmap lên màn hình (đích), thường nhận một trong các giá trị sau:

SRCCOPY - bitmap hiển thị trên màn hình hoàn hoàn như bitmap nguồn

SRCAND - thực hiện phép AND từng bit của bitmap nguồn với màn hình

SRCPAINT - thực hiện phép OR từng bit của bitmap nguồn với màn hình

SRCINVERT - thực hiện phép XOR từng bit của bitmap nguồn với màn hình

8.4. Huỷ DC ảo. Khi không dùng DC ảo nữa, cần huỷ nó như sau:

```
DeleteObject(hBitmap) ;
DeleteDC(hMemDC) ;
```

§9. CHƯƠNG TRÌNH MINH HOẠ

Chương trình sau đây minh hoạ các vấn đề về hiển thị văn bản lên vùng làm việc của cửa sổ chính. Chương trình gồm các chức năng sau:

+ Mở và huỷ cửa sổ con soạn thảo

+ Hiển thị văn bản đang soạn thảo (trong cửa sổ con) lên vùng làm việc cửa sổ chính

+ Trước khi hiển thị có thể:

- Chọn font, cỡ chữ (dùng hộp thoại Font)

- Chọn mầu chữ (dùng menu)

Chương trình trong thư mục **D:\WVC6\C-Win\Hien-VB**

//Tệp Resource.rc

```
#include "resource.h"
// Menu
HIEN_VB MENU DISCARDABLE
BEGIN
  POPUP "Hien thi VB"
  BEGIN
    MENUITEM "Chon Font",        IDM_FONT
    MENUITEM "Hien thi",         IDM_HT
    MENUITEM "Xoa CS",           IDM_XOA
  END
```

```
    POPUP "Chon mau"
    BEGIN
        MENUITEM "Red",                 IDM_RED
        MENUITEM "Green",               IDM_GREEN
        MENUITEM "Blue",                IDM_BLUE
    END
    POPUP "Soan thao VB"
    BEGIN
        MENUITEM "Mo CS soan thao",     IDM_MO_ST
        MENUITEM "Huy CS soan thao",    IDM_HUY_ST
    END
    MENUITEM "Ket thuc",                IDM_KT
    MENUITEM "Ve hinh tron",            IDM_VE
END
```

// Tệp Resource.h

```
#define IDM_FONT            40002
#define IDM_HT              40004
#define IDM_RED             40006
#define IDM_GREEN           40007
#define IDM_BLUE            40008
#define IDM_MO_ST           40009
#define IDM_HUY_ST          40010
#define IDM_KT              40011
#define IDM_XOA             40012
#define IDM_VE              40013
```

//Tệp Hien-vb.c

```
/*
    Hien_VB.C
        Minh hoa:
        + Cua so con soan thao
        + Chon Font, chon mau
        + Lay tung dong cua CS con ST cho hien len CS chinh
*/
#include "resource.h"
#include <windows.h>
```

```
#include <commdlg.h>
#include <stdio.h>
#include <string.h>
#define IDC_EDIT 3000
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
char szAppName[]="Hien_VB";  /* class name for the window      */
HINSTANCE hInst;
HWND hWndMain;
HWND hWndEdit=NULL;
RECT rr;
CHOOSEFONT cf;
LOGFONT lf;
HFONT hFont, hOldFont ;
COLORREF color;
char tg[80],vb[50][80];
int sodong=0;
int x,y,LineSpace;
TEXTMETRIC tm;
BOOL hienthi=FALSE;
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                   LPSTR lpszCmdLine, int nCmdShow)
{
 MSG       msg;        /* MSG structure to store your messages      */
 WNDCLASS  wndclass;   /* struct to define a window class           */
 hInst = hInstance;
 memset(&wndclass, 0x00, sizeof(WNDCLASS));
     /* load WNDCLASS with window's characteristics                 */
 wndclass.style = CS_HREDRAW | CS_VREDRAW |
                  CS_BYTEALIGNWINDOW;
 wndclass.lpfnWndProc = WndProc;
     /* Extra storage for Class and Window objects                  */
 wndclass.cbClsExtra = 0;
 wndclass.cbWndExtra = 0;
 wndclass.hInstance = hInst;
```

```
wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
        /* Create brush for erasing background                    */
wndclass.hbrBackground = GetStockObject(WHITE_BRUSH);
wndclass.lpszMenuName = szAppName;    /* Menu Name is App Name */
wndclass.lpszClassName = szAppName;    /* Class Name is App Name */
if(!RegisterClass(&wndclass))
{
   MessageBox(NULL,"Loi DK Lop CS",NULL,MB_OK);
        return 0;
}
        /* create application's Main window                       */
hWndMain = CreateWindow(
           szAppName,              /* Window class name    */
           "Chuong trinh soan thao",  /* Window's title       */
           WS_CAPTION    |         /* Title and Min/Max    */
           WS_SYSMENU    |         /* Add system menu box */
           WS_MINIMIZEBOX |        /* Add minimize box     */
           WS_MAXIMIZEBOX |        /* Add maximize box     */
           WS_THICKFRAME  |        /* thick sizeable frame  */
           WS_CLIPCHILDREN |       /* don't draw in child windows areas */
           WS_OVERLAPPED,
          CW_USEDEFAULT, CW_USEDEFAULT,   /*Use default X, Y*/
           CW_USEDEFAULT, CW_USEDEFAULT, /* Use default X, Y*/
           HWND_DESKTOP,           /* Parent window's handle    */
           NULL,                   /* Default to Class Menu     */
           hInst,                  /* Instance of window        */
           NULL);                  /* Create struct for WM_CREATE */
if(hWndMain == NULL)
  {
  MessageBox(NULL, "Loi XD CS", NULL, MB_ICONEXCLAMATION);
  return 0;
  }
 ShowWindow(hWndMain, nCmdShow); /* display main window          */
```

```
 while(GetMessage(&msg, NULL, 0, 0))  /* Until WM_QUIT message */
  {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
  }
 return msg.wParam;
}               /*  End of WinMain */
     LRESULT CALLBACK WndProc(HWND hWnd, UINT Message,
                   WPARAM wParam, LPARAM lParam)
{
HDC      hDC,hMemDC;          /* handle for the display device      */
HBITMAP hMemBitmap;
PAINTSTRUCT ps;               /* holds PAINT information            */
static HMENU hMenu;
int n,i;
 switch (Message)
  {
   case WM_COMMAND:
      /* The Windows messages for action bar and pulldown menu items */
      /* are processed here. */
      switch (wParam)
       {
       case IDM_MO_ST:
         GetClientRect(hWndMain,&rr);
         hWndEdit = CreateWindow("Edit",
               NULL,
               WS_CHILD | WS_VISIBLE |WS_VSCROLL |
               WS_HSCROLL | WS_BORDER |
               ES_AUTOVSCROLL | ES_AUTOHSCROLL |
               ES_LEFT |ES_NOHIDESEL | ES_MULTILINE ,
               0,rr.bottom/2, rr.right,   rr.bottom/2, hWndMain,
               (HMENU)IDC_EDIT, hInst, NULL);
          SetFocus(hWndEdit);   break;
       case IDM_HUY_ST:
       sodong=SendMessage(hWndEdit,EM_GETLINECOUNT,0,0L);
       for(i=0;i<sodong;++i)
        {
         vb[i][0]=80;  vb[i][1]=0;
```

```
  n=(int)SendMessage(hWndEdit,EM_GETLINE,i,(LPARAM)vb[i]);
  vb[i][n]=0;
 }
 DestroyWindow(hWndEdit);
 hWndEdit=NULL;
 break;
case IDM_FONT:
  memset(&cf,0,sizeof(CHOOSEFONT));
  cf.lStructSize = sizeof(CHOOSEFONT);
  cf.hwndOwner = hWnd;
  cf.lpLogFont = &lf;
  cf.Flags = CF_SCREENFONTS | CF_EFFECTS ;
  cf.rgbColors = RGB(0,255,255);
  cf.nFontType = SCREEN_FONTTYPE;
  ChooseFont(&cf);
  hFont = CreateFontIndirect(&lf);
   break;
case IDM_HT:
   hienthi=TRUE;
   InvalidateRect(hWnd,NULL,TRUE);   break;
case IDM_XOA:
  {
   RECT r; char tg[80]; HDC hDC;
   hDC=GetDC(hWnd);
   GetClientRect(hWnd,&r);
   sprintf(tg,"%d %d %d %d",r.left,r.top,r.right,r.bottom);
   SetMapMode(hDC,MM_HIENGLISH);
   GetClientRect(hWnd,&r);
   sprintf(tg,"%s %d %d %d %d",tg,r.left,r.top,r.right,r.bottom);
   MessageBox(hWnd,tg,"",MB_OK);
  }
  hienthi=FALSE;
  InvalidateRect(hWnd,NULL,TRUE);  break;
case IDM_RED:
   color = RGB(255,0,0);    break;
case IDM_GREEN:
   color = RGB(0,255,0);     break;
case IDM_BLUE:
```

```
            color = RGB(0,0,255);     break;
        case IDM_VE:
            hDC = GetDC(hWnd);
            hMemDC = CreateCompatibleDC(hDC);
            hMemBitmap = CreateCompatibleBitmap(hDC,50,50);
            SelectObject(hMemDC,hMemBitmap);
            Ellipse(hMemDC,0,0,49,49);
            BitBlt(hDC,200,200,50,50,hMemDC,0,0,SRCCOPY);
            DeleteObject(hMemBitmap);
            DeleteDC(hMemDC);
            ReleaseDC(hWnd,hDC);    break;
        case IDM_KT:
            PostMessage(hWnd,WM_CLOSE,0,0L);
        default:
            return DefWindowProc(hWnd, Message, wParam, lParam);
        }
    break;                  /* End of WM_COMMAND      */
case WM_CREATE:
    hMenu = GetMenu(hWnd);
    color = RGB(0,0,0);
    hFont = GetStockObject(SYSTEM_FONT);
    break;                  /*  End of WM_CREATE        */
case WM_MOVE:               /*  code for moving the window */
    break;
case WM_SIZE:               /*  code for sizing client area     */
    MoveWindow(hWndEdit,0,HIWORD(lParam)/2,LOWORD(lParam),
                                        HIWORD(lParam)/2,FALSE);
    break;                  /* End of WM_SIZE              */
case WM_PAINT:                      /* code for the window's client area */
    memset(&ps, 0x00, sizeof(PAINTSTRUCT));
    hDC = BeginPaint(hWnd, &ps);
                /* Included in case the background is not a pure color */
    SetBkMode(hDC, TRANSPARENT);
    if(hienthi)
      {
        SelectObject(hDC,hFont);  SetTextColor(hDC,color);
        GetTextMetrics(hDC,&tm);
        x=y=10;
```

```
            LineSpace = tm.tmHeight + tm.tmExternalLeading;
            if(hWndEdit!=NULL)
              {
              sodong=SendMessage(hWndEdit,EM_GETLINECOUNT,0,0L);
               for(i=0;i<sodong;++i)
                  {
                   tg[0]=80;  tg[1]=0;
              n=(int)SendMessage(hWndEdit,EM_GETLINE,i,(LPARAM)tg);
                   TextOut(hDC,x,y,tg,n); y += LineSpace;
                  }
              }
            else
               for(i=0;i<sodong;++i)
                  {
                    TextOut(hDC,x,y,vb[i],strlen(vb[i]));
                    y += LineSpace;
                  }
          }
        EndPaint(hWnd, &ps);  /* Inform Windows painting is complete  */
        break;                /*  End of WM_PAINT                     */
    case WM_CLOSE:            /* close the window                     */
         /* Destroy child windows, modeless dialogs, then, this window */
        if(hWndEdit!=NULL)
           DestroyWindow(hWndEdit);
        DestroyWindow(hWnd);
        if (hWnd == hWndMain)
           PostQuitMessage(0);          /* Quit the application       */
       break;
    default:
        /* For any message for which you don't specifically provide a  */
        /* service routine, you should return the message to Windows   */
        /* for default message processing                              */
     return DefWindowProc(hWnd, Message, wParam, lParam);
    }
    return 0L;
}   /* End of WndProc                          */
```

CHƯƠNG 7

ĐỒ HOẠ

§1. KHÁI NIỆM CƠ SỞ

Để vẽ các hình, cần sử dụng các khái niệm sau:

+ Màn hình là nơi hiển thị các hình
+ Hệ toạ độ và độ đo
+ Các hàm đồ hoạ GDI của Windows dùng để vẽ các hình
+ Các dụng cụ để vẽ là bút vẽ và chổi tô

1.1. Màn hình. Chúng ta có thể vẽ trên:

- Cửa sổ chính
- Cửa sổ con
- Hộp hội thoại
- Hộp soạn thảo (của hộp thoại)

Như vậy, màn hình (nơi sẽ xuất ra dữ liệu đồ hoạ) được hiểu là cửa sổ chính, cửa sổ con, hộp thoại hoặc hộp soạn thảo.

1.2. Hệ toạ độ và đơn vị đo. Khi vẽ các hình, đương nhiên phải sử dụng một hệ toạ độ và độ đo. Độ đo trong các hàm GDI là đơn vị logic. Thông thường chúng ta sử dụng hệ toạ độ mặc định và độ đo mặc định, vì vậy cần phải biết về chúng. Hệ toạ độ mặc định lấy điểm trên-trái màn hình làm điểm gốc. Trục hoành là đường nằm ngang theo chiều từ trái sang phải, trục tung là đường thẳng đứng theo chiều từ trên xuống dưới. Như vậy sẽ không có toạ độ âm, nói cách khác thì với mọi điểm (x,y) luôn luôn x>=0 và y>=0. Về đơn vị đo, thì mặc định mỗi đơn vị logic (dùng trong các hàm GDI) bằng một pixel. Chúng ta có dùng hàm SetMapMode để chọn một hệ toạ độ khác và một phép ánh xạ độ đo khác (xem §6)

1.3. Các hàm GDI. Các hàm GDI được dùng để xuất các dữ liệu đồ hoạ và văn bản ra thiết bị (trong chương này là màn hình). Các hàm này không làm việc trực tiếp với thiết bị mà làm việc với chỉ danh ngữ cảnh (DC) của thiết bị. Thư viện GDI sẽ dùng các trình điều khiển thiết bị (các tệp .DRV) để thực hiện việc xuất dữ liệu lên từng thiết bị cụ thể. Nhờ vậy các chương trình Windows được có tính độc lập với thiết bị (khi thay đổi thiết bị, thì chương trình gần như không phải sửa lại). Trong các mục §2 và §3 dưới đây sẽ giới thiệu một số hàm GDI dùng để vẽ các hình đồ hoạ.

1.4. Thủ tục xuất dữ liệu đã nói trong chương trước, gồm 3 bước: Lấy chỉ danh ngữ cảnh thiết bị, dùng các hàm GDI để xuất dữ liệu thông qua chỉ danh ngữ cảnh thiết bị và cuối cùng là giải phóng chỉ danh ngữ cảnh thiết bị. Các câu lệnh trong bước thứ 2 (bước chính) rõ ràng độc lập với thiết bị. Khi thay đổi thiết bị xuất, chỉ cần thay đổi vài dòng lệnh trong các bước 1 và 3.

1.5. Bút vẽ và chổi tô là các dụng cụ để vẽ. Bút vẽ cho biết mầu và độ rộng của nét vẽ. Chổi tô dùng để tô cho một miền, nó cho biết mầu tô và kiểu tô (xem §4)

§2. VẼ ĐƯỜNG VÀ VẼ HÌNH

Trong mục này trình bầy 10 hàm dùng để vẽ các đường và hình cơ bản. Khi vẽ cần dùng các khái niệm sau:

+ Vị trí hiện hành - là vị trí hiện tại trên màn hình mà một số hàm đồ hoạ thường bắt đầu vẽ từ đó.

+ Bút vẽ hiện hành - là bút đang được sử dụng để vẽ các đường

+ Chổi tô hiện hành - là chổi đang được sử dụng để tô các miền

+ Hoành độ, tung độ, độ dài (dùng trong các hàm đồ hoạ) - tính theo đơn vị logic

1. Hàm

BOOL MoveToEx(HDC hDC, int x, int y, LPPOINT lpPoint)

Công dụng: Thiết lập vị trí hiện hành mới và trả về vị trí hiện hành trước đó.

Các đối:

hDC là chỉ danh ngữ cảnh thiết bị

x, y là hoành độ và tung độ của vị trí hiện hành mới

lpPoint trỏ tới một biến kiểu POINT chứa toạ độ của vị trí hiện hành cũ

Giá trị của hàm:

Hàm trả về giá trị khác 0 nếu thành công và bằng 0 nếu có lỗi

Kiểu POINT được định nghĩa như sau

```
typedef struct tagPOINT
  {
    LONG x;
    LONG y;
  } POINT;
```

2. Hàm

BOOL LineTo(HDC hDC, int xEnd, int yEnd)

Công dụng: Vẽ một đoạn thẳng (theo bút hiện hành) từ vị trí hiện hành đến một vị trí xác định.

Các đối:

hDC là chỉ danh ngữ cảnh thiết bị

xEnd, yEnd là hoành độ và tung độ điểm cuối đoạn thẳng (điểm đầu là vị trí hiện hành)

Giá trị của hàm:

Hàm trả về giá trị khác 0 nếu thành công và bằng 0 nếu có lỗi

Nhận xét: Nếu thành công, thì (xEnd, yEnd) trở thành toạ độ của vị trí hiện hành mới.

3. Hàm

BOOL Polyline(HDC hDC, CONST POINT *lpPoint, int nPoints)

Công dụng: Vẽ một dãy đoạn thẳng (theo bút hiện hành) bằng cách nối các điểm chứa trong một mảng.

Các đối:

hDC là chỉ danh ngữ cảnh thiết bị

lpPoint trỏ tới một mảng kiểu POINT chứa toạ độ dãy điểm

nPoints là số điểm chứa trong mảng, cần lớn hơn hoặc bằng 2

Giá trị của hàm:

Hàm trả về giá trị khác 0 nếu thành công và bằng 0 nếu có lỗi

Nhận xét: Hàm không làm thay đổi vị trí hiện hành

4. Hàm

BOOL Polygon(HDC hDC, CONST POINT *lpPoint, int nPoints)

Công dụng: Vẽ hình đa giác, toạ độ các đỉnh của nó được chứa trong một mảng . Các cạnh được vẽ theo bút hiện hành, bên trong đa giác được tô theo chổi hiện hành.

Các đối:

hDC là chỉ danh ngữ cảnh thiết bị

lpPoint trỏ tới một mảng kiểu POINT chứa toạ độ các đỉnh đa giác

nPoints là số đỉnh chứa trong mảng, cần lớn hơn hoặc bằng 2

Giá trị của hàm:

Hàm trả về giá trị khác 0 nếu thành công và bằng 0 nếu có lỗi

Nhận xét: Hàm không làm thay đổi vị trí hiện hành

5. Hàm

BOOL Rectangle(HDC hDC, int nLeft, int nTop,int nRight,int nBottom)

Công dụng: Vẽ hình chữ nhật. Các cạnh được vẽ theo bút hiện hành, bên trong được tô theo chổi hiện hành.

Các đối:

hDC là chỉ danh ngữ cảnh thiết bị

nLeft, nTop là toạ độ của điểm góc trên-trái hình chữ nhật

nRight, nBoottm là toạ độ của điểm góc dưới-phải hình chữ nhật

Giá trị của hàm:

Hàm trả về giá trị khác 0 nếu thành công và bằng 0 nếu có lỗi

Nhận xét: Hàm không làm thay đổi vị trí hiện hành

6. Hàm

BOOL Ellipse(HDC hDC,int nLeft, int nTop, int nRight, int nBottom)

Công dụng: Vẽ hình elip, nội tiếp trong một hình chữ nhật (gọi là hình chữ nhật bao). Đường biên của hình elip được vẽ theo bút hiện hành, bên trong được tô theo chổi hiện hành.

Các đối:

hDC là chỉ danh ngữ cảnh thiết bị

nLeft, nTop là toạ độ của điểm góc trên-trái hình chữ nhật bao

nRight, nBoottm là toạ độ của điểm góc dưới-phải hình chữ nhật bao

Giá trị của hàm:

Hàm trả về giá trị khác 0 nếu thành công và bằng 0 nếu có lỗi

Nhận xét: Hàm không làm thay đổi vị trí hiện hành

7. Hàm

BOOL Arc(HDC hDC, int nLeft, int nTop, int nRight, int nBottom, int nXStart, int nYStart, int nXEnd, int nYEnd)

Công dụng: Vẽ cung elip (theo bút hiện hành), nó là một phần của đường elip nội tiếp trong hình chữ nhật (gọi là hình chữ nhật bao)

Các đối:

hDC là chỉ danh ngữ cảnh thiết bị

nLeft, nTop là toạ độ của điểm góc trên-trái hình chữ nhật bao

nRight, nBoottm là toạ độ của điểm góc dưới-phải hình chữ nhật bao

nXStart, nYStart là toạ độ điểm dùng để xác định điểm đầu, đó là giao của đường elip (nội tiếp hình chữ nhật) và nửa đường thẳng xuất phát từ tâm elip đi qua điểm (nXStart, nYStart)

nXEnd, nYEnd là toạ độ điểm dùng để xác định điểm cuối, đó là giao của đường elip và nửa đường thẳng xuất phát từ tâm elip đi qua điểm (nXSEnd, nYEnd)

Cách tạo cung elip:

Cung elip là một phần của đường elip đi từ điểm đầu đến điểm cuối theo chiều **ngược kim đồng hồ**.

Giá trị của hàm:

Hàm trả về giá trị khác 0 nếu thành công và bằng 0 nếu có lỗi

Nhận xét: Hàm không làm thay đổi vị trí hiện hành

8. Hàm

BOOL Pie(HDC hDC, int nLeft, int nTop, int nRight, int nBottom, int nXStart, int nYStart, int nXEnd, int nYEnd)

Công dụng: Vẽ hình quạt, nó là một phần của hình elip nội tiếp trong hình chữ nhật (gọi là hình chữ nhật bao). Đường biên của hình quạt được vẽ theo bút hiện hành, bên trong được tô theo chổi hiện hành.

Các đối:

hDC là chỉ danh ngữ cảnh thiết bị

nLeft, nTop là toạ độ của điểm góc trên-trái hình chữ nhật bao

nRight, nBoottm là toạ độ của điểm góc dưới-phải hình chữ nhật bao

nXStart, nYStart là toạ độ điểm dùng để xác định điểm đầu, đó là giao của đường elip (nội tiếp hình chữ nhật) và nửa đường thẳng xuất phát từ tâm elip đi qua điểm (nXStart, nYStart)

nXEnd, nYEnd là toạ độ điểm dùng để xác định điểm cuối, đó là giao của đường elip và nửa đường thẳng xuất phát từ tâm elip đi qua điểm (nXSEnd, nYEnd)

Cách tạo hình quạt:

Hình quạt là một phần của hình elip (nội tiếp hình chữ nhật bao) giới hạn bởi đường bán kính đầu, đường bán kính cuối và cung elip đi từ điểm đầu đến điểm cuối theo chiều **ngược kim đồng hồ**. Ở đây đường bán kính đầu là đường thẳng nối tâm elip với điểm đầu, còn đường bán kính cuối là đường thẳng nối tâm elip với điểm cuối.

Giá trị của hàm:

Hàm trả về giá trị khác 0 nếu thành công và bằng 0 nếu có lỗi

Nhận xét: Hàm không làm thay đổi vị trí hiện hành

9. Hàm

BOOL Chord(HDC hDC, int nLeft, int nTop, int nRight,
int nBottom, int nXStart,
int nYStart, int nXEnd,
int nYEnd)

Công dụng: Vẽ hình viên phân, nó là một phần của hình elip nội tiếp trong hình chữ nhật (gọi là hình chữ nhật bao). Hình viên phân sẽ là một mảnh của hình elip được cắt bởi một đường thẳng. Đường biên của hình viên phân được vẽ theo bút hiện hành, bên trong được tô theo chổi hiện hành.

Các đối:

hDC là chỉ danh ngữ cảnh thiết bị

nLeft, nTop là toạ độ của điểm góc trên-trái hình chữ nhật bao

nRight, nBoottm là toạ độ của điểm góc dưới-phải hình chữ nhật bao

nXStart, nYStart là toạ độ điểm dùng để xác định điểm đầu, đó là giao của đường elip (nội tiếp hình chữ nhật) và nửa đường thẳng xuất phát từ tâm elip đi qua điểm (nXStart, nYStart)

nXEnd, nYEnd là toạ độ điểm dùng để xác định điểm cuối, đó là giao của đường elip và nửa đường thẳng xuất phát từ tâm elip đi qua điểm (nXSEnd, nYEnd)

Cách tạo hình viên phân:

Hình viên phân là một phần của hình elip (nội tiếp hình chữ nhật bao) giới hạn bởi đoạn thẳng nối điểm đầu với điểm cuối và cung elip đi từ điểm đầu đến điểm cuối theo chiều **ngược kim đồng hồ**.

Giá trị của hàm:

Hàm trả về giá trị khác 0 nếu thành công và bằng 0 nếu có lỗi

Nhận xét: Hàm không làm thay đổi vị trí hiện hành

10. Hàm

BOOL RoundRect (HDC hDC,
int nLeft, int nTop,
int nRight, int nBottom,
int nWidth, int nHeight)

Công dụng: Vẽ hình chữ nhật góc tròn, nội tiếp trong một hình chữ nhật (gọi là hình chữ nhật bao). Đường biên của hình chữ nhật góc tròn được vẽ theo bút hiện hành, bên trong được tô theo chổi hiện hành.

Các đối:

hDC là chỉ danh ngữ cảnh thiết bị

nLeft, nTop là toạ độ của điểm góc trên-trái hình chữ nhật bao

nRight, nBoottm là toạ độ của điểm góc dưới-phải hình chữ nhật bao

nWidth, nHeight là chiều rộng và chiều cao của một cung elip, dùng để tạo đường cong tại các góc của hình chữ nhật.

Giá trị của hàm:

Hàm trả về giá trị khác 0 nếu thành công và bằng 0 nếu có lỗi

Nhận xét: Hàm không làm thay đổi vị trí hiện hành

§3. TÔ ĐIỂM, TÔ MIỀN

1. Hàm

COLORREF GetPixel (HDC hDC, int xPos, int yPos)

Công dụng: Cho biết mầu kiểu RGB tại một điểm xác định

Các đối:

hDC là chỉ danh ngữ cảnh thiết bị

xPos, yPos là toạ độ của điểm cần nhận mầu

Giá trị của hàm:

Hàm trả về giá trị mầu của điểm (xPos, yPos) nếu thành công và trả về giá trị CLR_INVALID nếu có lỗi.

Nhận xét: Điểm (xPos, yPos) cần nằm bên trong vùng hiển thị, nếu trái lại sẽ bị lỗi.

2. Hàm

COLORREF SetPixel (HDC hDC,
int xPos, int yPos,
COLOR crColor)

Công dụng: Tô mầu tại một điểm xác định

Các đối:

hDC là chỉ danh ngữ cảnh thiết bị

xPos, yPos là toạ độ của điểm cần tô mầu

crColor là mầu tô

Giá trị của hàm:

Hàm trả về giá trị mầu được tô nếu thành công và trả về giá trị -1 nếu có lỗi.

Nhận xét: Điểm (xPos, yPos) cần nằm bên trong vùng hiển thị, nếu trái lại sẽ bị lỗi.

3. Hàm

BOOL FloodFill (HDC hDC, int xStart, int yStart, COLOR crColor)

Công dụng: Tô miền bằng chổi tô hiện hành. Miền được xác định bằng mầu của đường bao.

Các đối:

hDC là chỉ danh ngữ cảnh thiết bị

xStart, yStart là toạ độ điểm bắt đầu tô (điểm gieo)

crColor là mầu của đường bao

Giá trị của hàm:

Hàm trả về giá trị khác 0 nếu thành công và giá trị 0 nếu có lỗi.

Nhận xét: Lỗi có thể xẩy ra trong các trường hợp:

+ Mầu của điểm gieo trùng với mầu đường bao

+ Điểm gieo nằm ngoài vùng hiển thị

Khi đó việc tô mầu sẽ không thực hiện.

4. Hàm

BOOL ExtFloodFill (HDC hDC,
int xStart, int yStart,
COLOR crColor, UINT uFillType)

Công dụng: Tô miền bằng chổi tô hiện hành. Miền được xác định bằng mầu của đường bao hoặc xác định bằng mầu của miền.

Các đối:

hDC là chỉ danh ngữ cảnh thiết bị

xStart, yStart là toạ độ điểm bắt đầu tô (điểm gieo)

crColor là mầu của đường viền hoặc mầu của miền

uFillType xác định kiểu tô, nó có thể nhận một trong 2 giá trị:

FLOODFILLBORDER-Tô miền giới hạn bằng một đường bao có mầu crColor

FLOODFILLSURFACE-Tô miền có mầu crColor

Giá trị của hàm:

Hàm trả về giá trị khác 0 nếu thành công và giá trị 0 nếu có lỗi.

Nhận xét: Lỗi có thể xảy ra trong các trường hợp:

+ Mầu của điểm gieo trùng với mầu đường viền
  (khi uFillType= FLOODFILLBORDER)
+ Mầu của điểm gieo không trùng với mầu của miền
  (khi uFillType= FLOODFILLSURFACE)
+ Điểm gieo nằm ngoài vùng hiển thị

Khi đó việc tô mầu sẽ không thực hiện.

§4. MẦU, BÚT VẼ, CHỔI TÔ

Trong các hàm vẽ đường và hình cơ bản (xem §2) đều sử dụng đến bút vẽ hiện hành và chổi tô hiện hành. Để thay đổi bút vẽ hiện hành cần 2 động tác: Lấy chỉ danh bút vẽ mới, sau đó gắn nó với DC thiết bị. Tương tự để chọn chổi tô mới cũng cần 2 công đoạn: Lấy chỉ danh chổi tô mới và gắn nó với DC thiết bị. Mỗi bút vẽ và chổi tô có một mầu nào đó. Trong mục này sẽ trình bầy các hàm dùng để lấy chỉ danh bút vẽ, chỉ danh chổi tô và gắn chúng với DC thiết bị.

4.1. Cách biểu diễn mầu

+ Mầu là sự tổ hợp của 3 mầu cơ bản: Red (đỏ), Green(xanh lá cây), Blue(xanh lam). Vì vậy giá trị mầu thường gọi là : mầu RGB (RGB color) hay giá trị mầu RGB (RGB color value).

+ Để biểu diễn mầu RGB, dùng kiểu COLORREF là một số nguyên 32 bít trong đó 8 bit chứa cường độ (intensity) của Red, 8 bit chứa cường độ (intensity) của Green, 8 bit chứa cường độ (intensity) của Blue, 8 bit không dùng có giá trị bằng 0. Nói cụ thể hơn một giá trị kiểu COLORREF có thể viết theo hệ Hexa (hệ 16) như sau:

  0x00bbggrr

+ Để có được một mầu RGB, có thể dùng hàm:

  COLORREF RGB(BYTE bRed, bGreen, bBlue)

Các đối:

bRed là cường độ của mầu Red

bGreen là cường độ của mầu Green

bBlue là cường độ của mầu Blue

Giá trị hàm:

Hàm trả về một giá trị mầu RGB

+ Để tách các mầu cơ bản từ mộ mầu RGB, có thể dùng các hàm:

```
BYTE GetRValue(COLORREF rgbColor) // Tách mầu Red
BYTE GetGValue(COLORREF rgbColor) // Tách mầu Green
BYTE GetBValue(COLORREF rgbColor) // Tách mầu Blue
```

+ Dưới đây là một số mầu RGB thường dùng:

RGB(0, 0, 0) - đen	RGB(255, 255, 255) - trắng
RGB(255, 0, 0) - đỏ	RGB(255, 0, 128) - đỏ nhạt
RGB(255, 0, 255) - hồng	RGB(255, 128, 255) - hồng nhạt
RGB(128, 0, 128) - tím	RGB(128, 0, 255) - tím nhạt
RGB(255, 255, 0) - vàng	RGB(255, 255, 128) - vàng nhạt
RGB(0, 0, 255) - x. lam	RGB(0, 0, 128) - x. lam nhạt
RGB(0, 255, 0) - x. lá cây	RGB(128, 255, 128) - x. lá cây nhạt
RGB(0, 255, 255) - x. lơ	RGB(128, 255, 255) - x. lơ nhạt
RGB(128, 128, 128) - xám	RGB(192, 192, 192) - xám nhạt
RGB(128, 0, 0) - nâu	

4.2. Để nhận chỉ danh bút vẽ chuẩn của Windows, có thể dùng hàm GetStockObject theo mẫu sau:

```
HPEN hPen;
hPen = (HPEN) GetStockObject(nPenObject) ;
```

Trong đó tham số nPenObject có thể nhận các giá trị:

BLACK_PEN - Bút đen

WHITE_PEN - Bút trắng

NULL_PEN - Bút rỗng

Nhận xét: Bút rỗng dùng để vẽ các hình mở (không có đường viền bao quanh)

4.3. Để nhận chỉ danh chổi tô chuẩn của Windows, có thể dùng hàm GetStockObject theo mẫu sau:

```
HBRUSH hBrush;
hBrush = (HBRUSH) GetStockObject(nBrushObject) ;
```

Trong đó tham số nBrushObject có thể nhận các giá trị:

BLACK_BRUSH - chổi đen
WHITE_BRUSH - chổi trắng
LTGRAY_BRUSH - chổi xám nhạt
DKGRAY _BRUSH - chổi xám sẫm
NULL_BRUSH - Chổi rỗng

Nhận xét: Chổi rỗng dùng để vẽ các đường kín bao quanh một hình nào đó, như đường elip, đường tròn, đường chữ nhật, đường đa giác, ...

4.4. Để tạo và lấy chỉ danh bút vẽ mới, cần dùng hàm:

```
HPEN CreatePen(int nPenStyle,
               int nPenWidth,
               COLORREF cfPenColor)
```

Các đối:

nPenStyle là kiểu bút, có có thể nhận các giá trị sau:

PS_SOLID - Vẽ bằng đường nét liền
PS_NULL - Không vẽ (Bút rỗng)
PS_DOT - Vẽ bằng các dấu chấm
PS_DASH - Vẽ bằng các dấu gạch ngang
PS_DASHDOT - Vẽ bằng các cặp 2 dấu: Gạch ngang và chấm
PS_DASHDOTDOT -Vẽ bằng các bộ 3 dấu: Gạch ngang, chấm và chấm
PS_INSIDEFRAME - Vẽ bằng đường nét liền trong vùng khép kín

nPenWidth là độ rộng của đường vẽ, tính theo đơn vị logic.

Chú ý: là chỉ có đường nét liền mới có thể có độ rộng lơn hơn một đơn vị.

nPenColor là mầu của đường vẽ

Giá trị hàm

Nếu thành công, hàm trả về chỉ danh của bút mới được tạo. Nếu có lỗi hàm trả về NULL

4.5. Để tạo và lấy chỉ danh một chổi tô kín, cần dùng hàm:

```
HBRUSH CreateSolidBrush(COLORREF cfBrushColor)
```

Các đối:

nBrushColor là mầu của chổi tô (mầu tô). Cách tô ở đây là tô kín.

Giá trị hàm

Nếu thành công, hàm trả về chỉ danh của chổi tô mới được tạo. Nếu có lỗi hàm trả về NULL

4.6. Để tạo và lấy chỉ danh một chổi tô không kín, cần dùng hàm:

```
HBRUSH CreateHatchBrush(int nBrushStyle,
                        COLORREF cfBrushColor)
```

Các đối:

nBrushColor là mầu của chổi tô (mầu tô)

nBrushStyle xác định kiểu tô, nó có thể nhận một trong các giá trị sau:

HS_HORIZONTAL - Tô bằng các đường nằm ngang

HS_VERTICAL - Tô bằng các đường thẳng đứng

HS_FDIAGONAL - Tô bằng các đường chéo chính: \

HS_BDIAGONAL - Tô bằng các đường chéo phụ: /

HS_HCROSS - Tô bằng các ô vuông (tạo bổi các đường ngang và đứng)

HS_DIAGROSS - Tô bằng các ô chéo (tạo bổi các chéo chính và chéo phụ)

Giá trị hàm

Nếu thành công, hàm trả về chỉ danh của chổi tô mới được tạo. Nếu có lỗi hàm trả về NULL

4.7. Để gắn bút vẽ và chổi tô vào DC thiết bị, có thể dùng hàm SelectObject theo mẫu sau:

```
HPEN hOldPen, hPen ;
HBRUSH hOldBrush, hBrush ;
                                    // Gắn hPen vào DC thiết bị và
                                    // lưu chỉ danh bút hiện hành
hOldPen = SelectObject(hDC, hPen); // hPen trở thành bút hiện hành
                                    // Gắn hBrush vào DC thiết bị và
                                    // lưu chỉ danh chổi hiện hành
hOldBrush = SelectObject(hDC, hBrush);          // hBrush trở thành chổi hiện hành
```

4.8. Để huỷ bút vẽ, chổi tô, có thể dùng hàm DeleteObject theo mẫu:

```
DeleteObject(hPen);    // Huỷ bút hPen
DeleteObject(hBrush);  // Huỷ chổi hBrush
```

Nguyên mẫu của hàm:

BOOL DeleteObject(HGDIOBJ hObject)

trong đó:

hObject là chỉ danh của một đối tượng đồ hoạ cần huỷ

Hàm trả về giá trị khác 0 nếu thành công và bằng 0 nếu có lỗi.

4.9. Để tạo và sử dụng bút vẽ, chổi tô, có thể dùng mẫu sau:

```
    // Khai báo các biến để chứa chỉ danh bút vẽ và chổi tô
HPEN  hOldPen, hPen ;
HBRUSH  hOldBrush, hBrush ;
    // Tạo và lấy chỉ danh bút vẽ, chổi tô mới
hPen = CreatePen(PS_SOLID, 2, RGB(255,0,0)) ; // Bút đỏ, nét liền, rộng 2
hBrush = CreateSolidBrush(RGB(0, 0, 255) ); //Chổi tô kín, mầu xanh BLUE
                                    // Gắn bút mới, chổi mới vào DC
                                    // thiết bị, lưu bút, chổi hiện hành
hOldPen = SelectObject(hDC, hPen);     // hPen trở thành bút hiện hành
hOldBrush = SelectObject(hDC, hBrush);// hBrush trở thành chổi hiện hành
                                     // Xử dụng bút mới, chổi mới để vẽ
Rectangle(hDC, 20, 20, 200, 200);
Ellipse(hDC, 300, 20, 400, 200);
   // Khôi phục bút và chổi cũ
SelectObject(hDC, hOldPen);
SelectObject(hDC, hOldBrush);
   // Huỷ bút và chổi tạo bên trên
DeleteObject(hPen);
DeleteObject(hBrush);
```

§5. SỬ DỤNG HỘP THOẠI CHỌN MẦU

5.1. Hộp thoại mầu (Colors) gồm:

+ Bảng 48 mầu cơ sở đã có sẵn và không thay đổi được (gọi là Basic colors)
+ Bảng16 mầu tự tạo (gọi là Custom colors)
+ Các nút OK và Cancel
+ Nút lệnh Add to Custom Colors dùng để tạo một mầu (bằng cách điền giá trị vào 3 ô RED, GREEN, BLUE) và bổ sung vào bảng Custom colors

5.2. Để mở hộp thoại Colors cần dùng hàm:

BOOL ChooseColor(LPCHOOSECOLOR lpcc) ;

Đối: lpcc trỏ tới một biến cấu trúc kiểu CHOOSECOLOR, chứa các thông tin được dùng để khởi tạo và mở hộp thoại Colors.

Khi thoát ra khỏi hàm: Thuộc tính rgbResult của cấu trúc sẽ chứa mầu được chọn hoặc chứa mầu mặc định được khởi tạo.

Giá trị của hàm:

+ Nếu người sử dụng chọn một mầu trong bảng Basic colors hoặc trong bảng Custom colors rồi bấm nút OK, thì hàm trả về giá trị khác 0 và thuộc tính rgbResult của cấu trúc sẽ chứa giá trị RGB của mầu được chọn.

+ Nếu người sử dụng bấm phím Cancel hoặc đóng hộp thoại Colors, thì hàm trả về giá trị 0 và thuộc tính rgbResult của cấu trúc sẽ chứa giá trị RGB được khởi tạo ban đầu (trước khi gọi hàm ChooseColor)

5.3. Cấu trúc CHOOSECOLOR gồm nhiều thuộc tính dùng để chứa các thông tin dùng để khởi tạo trạng thái ban đầu cho hộp thoại Colors. Dưới đây là các thuộc tính thường dùng:

DWORD lStructSize - chứa độ dài theo byte của cấu trúc

HWND hwndOwner - chứa chỉ danh cửa sổ cha (cửa sổ sở hữu hộp thoại Colors)

COLORREF rgbResult - chứa mầu được chọn bởi người sử dụng

COLORREF *lpCustColors - trỏ tới mảng chứa 16 giá trị RGB dùng để khởi tạo cho bảng Custom colors

DWORD Flags - cho biết có hiển thị nút Add to Custom Colors hay không?

Nếu Flags = CC_PREVENTFULLOPEN thì không hiển thị

Nếu Flags = CC_FULLOPEN thì hiển thị

§6. HỆ TOẠ ĐỘ VÀ ĐƠN VỊ ĐO CHUẨN

6.1. Hệ toạ độ chuẩn dùng để xuất dữ liệu bao gồm:

a. Kích thước của hệ toạ độ là cửa sổ (vùng làm việc cửa sổ). Nói cách khác có thể xuất các dữ liệu đồ hoạ trên toàn bộ cửa sổ.

b. Gốc toạ độ là điểm có toạ độ (0, 0). Mặc định, gốc toạ độ là điểm góc trên-trái vùng nhìn. Để thiết lập gốc toạ độ mới, cần dùng hàm: SetViewportOrgEx (xem dưới đây).

c. Chiều tăng của hoành độ x và tung độ y. Hoành độ x luôn luôn tăng từ trái sang phải (x dương bên phải trục tung). Mặc định thì y tăng từ trên xuống dưới (y dương bên dưới trục hoành). Dùng hàm SetMapMode (xem bên dưới) có thể thay đổi chiều tăng của y.

6.2. Đơn vị đo.

Trong các hàm đồ hoạ dùng đơn vi đo toán học, gọi là đơn vị logic. Mặc định, mỗi đơn vị logic được qui đổi thành một pixel. Khi dùng hàm SetMapMode dưới đây, ta có thể chọn các phép quy đổi khác, như quy đổi một đơn vị logic thành 0.01 mm hay thành 0.001 inch.

6.3. Hàm SetMapMode

Có thể dùng SetMapMode để chọn phép quy đổi độ đo và chọn chiều tăng của trục y.

Nguyên mẫu của hàm như sau:

int SetMapMode(HDC hDC, int nMapMode)

Các đối:

+ hDC là chỉ danh ngữ cảnh thiết bị (cụ thể ở đây là màn hình)

+ nMapMode xác định phép quy đổi độ đo và chiều tăng trục y, nMapMode có thể nhận các giá trị sau:

MM_TEXT - Mỗi đv logic quy đổi thành một pixel, y dương bên dưới ox

MM_HIENGLISH - Mỗi đv logic quy đổi thành 0.001 inch, y dương bên trên ox

MM_HIMETRIC - Mỗi đv logic quy đổi thành 0.01 mm, y dương bên trên ox

MM_LOENGLISH - Mỗi đv logic quy đổi thành 0.01 inch, y dương bên trên ox

MM_LOMETRIC - Mỗi đv logic quy đổi thành 0.1 mm, y dương bên trên ox

MM_TWIPS - Mỗi đv logic quy đổi thành 1/1440 inch, y dương bên trên ox

MM_ANISOTROPIC - Xét trong mục tiếp theo

MM_ISOTROPIC - Xét trong mục tiếp theo

Ghi chú: Chế độ ánh xạ mặc định là MM_TEXT

Giá trị hàm:

Nếu thành công hàm trả về một giá trị nguyên xác định Mapmode (chế độ ánh xạ) đang sử dụng.

Nếu có lỗi hàm trả về giá trị 0.

6.4. Hàm SetViewportOrgEx

Hàm SetViewportOrgEx dùng để thiết lập điểm gốc toạ độ mới, nguyên mẫu của nó như sau:

```
BOOL SetViewportOrgEx( HDC hDC,
                       int x, int y,
                       LPPOINT lpPoint)
```

Các đối:

+ hDC là chỉ danh ngữ cảnh thiết bị (cụ thể ở đây là màn hình)

+ x, y là toạ độ điểm tâm mới của hệ toạ độ. Giá trị của x, y tính theo pixel trong hệ toạ độ mặc định (gốc là điểm trên-trái cửa sổ, x tăng từ trái sang phải, y tăng từ trên xuống dưới)

+ lpPoint trỏ tới một cấu trúc kiểu POINT dùng để lưu toạ độ của điểm tâm hiện đang sử dụng. Nếu lpPoint = NULL thì không lưu lại toạ độ tâm hiện thời.

Giá trị hàm:

Nếu thành công hàm trả về một giá trị khác 0.

Nếu có lỗi hàm trả về giá trị 0.

6.5. Ví dụ Xét các lệnh:

```
int cxClient, cyClient ;
RECT r;
HDC hDC;
GetClientRect(hWnd, &r);
cxClient =  r.right ;  cyClient = r.bottom ;
hDC = GetDC(hWnd);
SetViewportOrgEx(hDC, cxClient /2, cxClient /2, NULL);
```

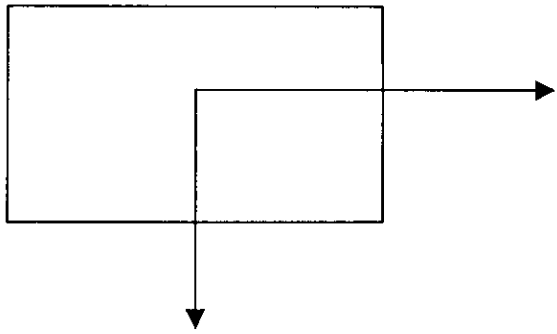
Sau khi thực hiện các lệnh trên, chúng ta sẽ nhận được hệ toạ độ sau:

+ Gốc toạ độ tại giữa màn hình. Nói cách khác điểm giữa màn hình có toạ độ (0, 0)

+ x tăng từ trái sang phải, y tăng từ trên xuống dưới

+ Điểm góc trên-trái có toạ độ (-cxClent/2, -cyClient/2)

+ Điểm góc dưới-phải có toạ độ (cxClent/2, cyClient/2)



§7. HỆ TOẠ ĐỘ VÀ ĐƠN VỊ ĐO TỰ ĐẶT

Để xác định một hệ toạ độ và đơn vị đo tự đặt, cần dùng các hàm sau:

7.1. Dùng hàm

int SetMapMode(HDC hDC, int nMapMode)

với 2 giá trị sau của nMapMode:

a. MM_ANISOTROPIC

b. MM_ISOTROPIC

Ý nghĩa:

+ Mỗi đơn vị logic được đổi thành một đơn vị do người lập trình tự đặt.

+ Trường hợp a, hệ số quy đổi là khác nhau đối với hoành độ và tung độ

+ Trường hợp b, hệ số quy đổi là như nhau đối với hoành độ và tung độ

+ Để xác định đơn vị đo và chiều tăng của các trục, cần dùng 2 hàm SetWindowExtEx và SetViewportExtEx

7.2. Để xác định bề ngang và bề dọc cửa sổ theo độ đo mới, dùng hàm:

BOOL SetWindowExtEx(HDC hDC,
int nXExtent,
int nYExtent,
LPSIZE lpSize)

Các đối:

+ hDC là chỉ danh ngữ cảnh thiết bị (cụ thể ở đây là màn hình)

+ nXExtent là kích thước bề ngang cửa sổ theo đơn vị đo mới

+ nYExtent là kích thước bề dọc cửa sổ theo đơn vị đo mới

+ lpSIZE trỏ tới một cấu trúc kiểu SIZE dùng để lưu kích thước các bề ngang, dọc của cửa sổ theo đơn vị đo đang dùng. Nếu lpSize = NULL thì không lưu lại gì.

Giá trị hàm:

Nếu thành công hàm trả về một giá trị khác 0.

Nếu có lỗi hàm trả về giá trị 0.

Chi chú:

a. Nếu nMapMode (trong hàm SetMapMode) bằng MM_ANISOTROPIC thì:

Mỗi đơn vị logic của hoành độ x quy đổi thành cxClient/nXExtent pixel

Mỗi đơn vị logic của tung độ y quy đổi thành 0cyClient/nYExtent pixel

b. Nếu nMapMode (trong hàm SetMapMode) bằng MM_ISOTROPIC thì mỗi đơn vị logic (của cả hoành độ x và tung độ y) đều được quy đổi thành:

min(cxClient/nXExtent, cyClient/nYExtent) pixel

Ở đây cxClient và cyClient là kích thước bề ngang và bề dọc cửa sổ tính theo pixel, chúng có thể xác định nhờ hàm GetClientRect.

7.3. Dùng hàm SetViewportExtEx để xác định độ lớn vùng nhìn

BOOL SetViewportExtEx (HDC hDC,
int cxViewport,
int cyViewport,
LPSIZE lpSize)

Các đối:

+ hDC là chỉ danh ngữ cảnh thiết bị (cụ thể ở đây là màn hình)

+ cxViewport là kích thước bề ngang vùng nhìn theo pixel (đơn vị thiết bị)

+ cyViewport là kích thước bề dọc vùng nhìn theo pixel (đơn vị thiết bị)

+ lpSIZE trỏ tới một cấu trúc kiểu SIZE dùng để lưu kích thước các bề ngang, dọc của vùng nhìn đang dùng. Nếu lpSize = NULL thì không lưu lại gì.

Giá trị hàm:

Nếu thành công hàm trả về một giá trị khác 0.

Nếu có lỗi hàm trả về giá trị 0.

Ghi chú:

a. Khi cyViewport > 0, thì chiều tăng của y là từ trên xuống dưới

Khi cyViewport < 0, thì chiều tăng của y là từ dưới lên trên

b. Kích thước thực sự của cửa sổ (theo đơn vị mới) phụ thuộc vào độ lớn vùng nhìn. Cụ thể như sau:

+ Nếu vùng nhìn bằng cửa sổ, thì kích thước thực sự cửa sổ sẽ bằng kích thước xác định trong hàm SetWindowExtEx (xem các ví dụ dưới đây)

+ Nếu vùng nhìn chỉ bằng nửa cửa sổ, thì kích thước thực sự cửa sổ sẽ bằng 2 lần kích thước xác định trong hàm SetWindowExtEx (xem các ví dụ dưới đây)

7.4. Hệ toạ độ dùng cho các hàm đồ hoạ

+ Để xác định hệ toạ độ có gốc là điểm trên-trái cửa sổ, cần dùng 3 hàm:

SetMapMode

SetWindowExtEx

SetViewportExtEx

+ Để thay đổi gốc toạ độ cần dùng thêm hàm:

SetViewportOrgEx

Các ví dụ dưới đây sẽ minh hoạ cách dùng các hàm nêu trên.

7.5. Các ví dụ về chế độ ánh xạ MM_ANISOTROPIC

Trong các ví dụ dưới đây, ký hiệu cx là kích thước bề ngang cửa sổ, cy là kích thước bề dọc cửa sổ, cả hai đều tính theo pixel (có thể nhận cx, cy bằng hàm GetClientRect)

VD1. Các lệnh:

```
SetMapMode(hDC, MM_ANISOTROPIC);
SetWindowExtEx(hDC, 5000, 5000, NULL);
SetViewportExtEx(hDC, cx, cy, NULL);
```

sẽ cho hệ toạ độ:

+ Gốc là điểm trên-trái cửa sổ
+ Điểm dưới phải có toạ độ (5000, 5000);

VD2. Các lệnh:

```
SetMapMode(hDC, MM_ANISOTROPIC);
SetWindowExtEx(hDC, 5000, 5000, NULL);
SetViewportExtEx(hDC, cx, -cy, NULL);
```

sẽ cho hệ toạ độ:

+ Gốc là điểm trên-trái cửa sổ
+ Điểm dưới phải có toạ độ (5000, -5000);

VD3. Các lệnh:

```
SetMapMode(hDC, MM_ANISOTROPIC);
SetWindowExtEx(hDC, 5000, 5000, NULL);
SetViewportExtEx(hDC, cx/2, -cy/2, NULL);
```

sẽ cho hệ toạ độ:

+ Gốc là điểm trên-trái cửa sổ
+ Điểm dưới phải có toạ độ (10000, -10000);

VD4. Các lệnh:

```
SetMapMode(hDC, MM_ANISOTROPIC);
SetWindowExtEx(hDC, 5000, 5000, NULL);
SetViewportExtEx(hDC, cx, cy, NULL);
SetViewportOrgEx(hDC, cx/2, cy/2, NULL);
```

sẽ cho hệ toạ độ:

+ Gốc là điểm giữa cửa sổ
+ Điểm trên-trái có tạo độ (-2500, -2500)
+ Điểm dưới-phải có toạ độ (2500, 2500);

VD5. Các lệnh:

```
SetMapMode(hDC, MM_ANISOTROPIC);
SetWindowExtEx(hDC, 5000, 5000, NULL);
SetViewportExtEx(hDC, cx, -cy, NULL);
SetViewportOrgEx(hDC, cx/2, cy/2, NULL);
```

sẽ cho hệ toạ độ:

+ Gốc là điểm giữa cửa sổ
+ Điểm trên-trái có tạo độ (-2500, 2500)
+ Điểm dưới-phải có toạ độ (2500, -2500);

VD6. Các lệnh:

```
SetMapMode(hDC, MM_ANISOTROPIC);
SetWindowExtEx(hDC, 5000, 5000, NULL);
SetViewportExtEx(hDC, cx/2, -cy/2, NULL);
SetViewportOrgEx(hDC, cx/2, cy/2, NULL);
```

sẽ cho hệ toạ độ:

+ Gốc là điểm giữa cửa sổ
+ Điểm trên-trái có tạo độ (-5000, 5000)
+ Điểm dưới-phải có toạ độ (5000, -5000);

7.6. Các ví dụ về chế độ ánh xạ MM_ISOTROPIC

Cũng như trong điểm trên, ở đây ký hiệu cx là kích thước bề ngang cửa sổ, cy là kích thước bề dọc cửa sổ, cả hai đều tính theo pixel. Ngoài ra, giả sử cx > cy. Khi đó hệ số quy đổi được tính theo trục y.

VD1. Các lệnh:

```
SetMapMode(hDC, MM_ISOTROPIC);
SetWindowExtEx(hDC, 5000, 5000, NULL);
SetViewportExtEx(hDC, cx, cy, NULL);
```

sẽ cho hệ toạ độ:

+ Gốc là điểm trên-trái cửa sổ
+ Điểm dưới phải có toạ độ ((cx/cy)*5000, 5000);

VD2. Các lệnh:

```
SetMapMode(hDC, MM_ISOTROPIC);
SetWindowExtEx(hDC, 5000, 5000, NULL);
SetViewportExtEx(hDC, cx, -cy, NULL);
```

sẽ cho hệ toạ độ:

+ Gốc là điểm trên-trái cửa sổ
+ Điểm dưới phải có toạ độ ((cx/cy)*5000, -5000);

VD3. Các lệnh:

```
SetMapMode(hDC, MM_ISOTROPIC);
SetWindowExtEx(hDC, 5000, 5000, NULL);
SetViewportExtEx(hDC, cx/2, -cy/2, NULL);
```

sẽ cho hệ toạ độ:

+ Gốc là điểm trên-trái cửa sổ
+ Điểm dưới phải có toạ độ ((cx/cy)*10000, -10000);

VD4. Các lệnh:

```
SetMapMode(hDC, MM_ISOTROPIC);
SetWindowExtEx(hDC, 5000, 5000, NULL);
SetViewportExtEx(hDC, cx, cy, NULL);
SetViewportOrgEx(hDC, cx/2, cy/2, NULL);
```

sẽ cho hệ toạ độ:

+ Gốc là điểm giữa cửa sổ
+ Điểm trên-trái có tạo độ (-(cx/cy)*2500, -2500)
+ Điểm dưới-phải có toạ độ ((cx/cy)*2500, 2500);

VD5. Các lệnh:

```
SetMapMode(hDC, MM_ISOTROPIC);
SetWindowExtEx(hDC, 5000, 5000, NULL);
SetViewportExtEx(hDC, cx, -cy, NULL);
SetViewportOrgEx(hDC, cx/2, cy/2, NULL);
```

sẽ cho hệ toạ độ:

+ Gốc là điểm giữa cửa sổ
+ Điểm trên-trái có tạo độ (-(cx/cy)*2500, 2500)
+ Điểm dưới-phải có toạ độ ((cx/cy)*2500, -2500);

VD6. Các lệnh:

```
SetMapMode(hDC, MM_ISOTROPIC);
SetWindowExtEx(hDC, 5000, 5000, NULL);
SetViewportExtEx(hDC, cx/2, -cy/2, NULL);
SetViewportOrgEx(hDC, cx/2, cy/2, NULL);
```

sẽ cho hệ toạ độ:

+ Gốc là điểm giữa cửa sổ
+ Điểm trên-trái có tạo độ (-(cx/cy)*5000, 5000)
+ Điểm dưới-phải có toạ độ ((cx/cy)*5000, -5000);