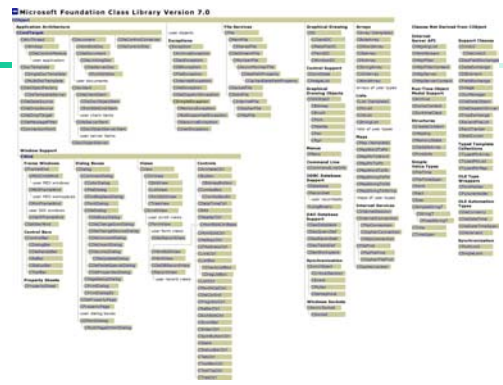
Costruzione di Interfacce
Lezione 16
Primi passi MFC

cignoni@iei.pi.cnr.it
http://vcg.iei.pi.cnr.it/~cignoni

Introduzione

- Microsoft Foundation Class Library (MFC) è un "application framework" per scrivere applicazioni per Windows.
- Le MFC sono implementate come un insieme di classi C++, molte delle quali rappresentano oggetti comuni come documenti, finestre, dialog box ecc.
- Come tutti i framework per programmare sotto windows nasconde bene come funziona davvero windows.



MFC

- Le MFC mettono a disposizione varie classi per rappresentare la struttura di una applicazione:
- Concetti fondamentali rappresentati tramite oggetti
  - applicazione
  - finestra
  - documento
  - vista
- Un'applicazione è un'istanza di un oggetto.

CWinApp

- La CWinApp class è la classe base da cui si deriva la classe per rappresentare un applicazione.
- Si definiscono funzioni membro per inizializzare la propria applicazione (ed ogni sua istanza) e per i vari compiti propri dell'applicazione.

CWinApp

- La classe **CWinApp** incapsula, l'inizializzazione, l'esecuzione e la terminazione di un'applicazione in ambiente Windows
- Un'applicazione deve avere uno (ed un solo) oggetto di una classe derivata da **CWinApp** (application object)
- Un application object ha membri per inizializzare e far girare l'applicazione stessa
- L'oggetto è costruito prima che le finestre dell'app. vengano create.

Note storiche: WinMain

- Sequenza di esecuzione di un generico programma windows a basso livello…
- Quando parte un applicazione windows l'entry point è una funzione chiamata winmain, (equivalente del main di un programma c, o c++ in un ambiente windows console)
-
```
int WINAPI WinMain(
  HINSTANCE hInstance,     // handle to current instance
  HINSTANCE hPrevInstance, // handle to previous instance (ora sempre 0)
  LPSTR lpCmdLine,         // command line
  int nCmdShow             // show state );
```

WinMain dovrebbe inizializzare l'applicazione, mostrare la finestra principale ed entrare nel ciclo principale (main loop) di raccolta/smistamento dei messaggi del s.o. che continua in eterno finchè non arriva un messaggio **WM_QUIT**

Note storiche: Handle

- in pratica sono
  - O puntatori
  - O offset relativo ad una qualche tabella del sistema operativo,
- Aggiungono un livello di indirezione per accedere a vari oggetti in qualche modo relati al s.o.
- Di solito le variabili che sono handle a qualcosa iniziano sempre per 'H' (hWnd, hBrush ecc)
- Note:
  - Tutta colpa di windows30
  - Una bella spiegazione (utile soprattutto per i curiosi) la trovate sull'help di dot.net cercando "**Give Me a Handle, and I'll Show You an Object**" (del 1993!)

Note storiche: **Messaggi**

- La gestione degli eventi da parte di un applicazione è fatta tramite messaggi:
- Ogni finestra ha, obbligatoriamente, una callback che serve a gestire i messaggi a lei diretti
- Il loop classico di un app win è quindi qualcosa del tipo:
-
```
while( GetMessage( &msg, NULL, 0, 0 ) ) {
    DispatchMessage( &msg );
}
```
- Dove GetMessage prende un messaggio dalla coda e DispatchMessage lo manda alla callback della finestra giusta

Note storiche: multitasking

- Perchè i programmi di windows hanno questa #@$%# struttura?
  - Cooperative multitasking,
    - I processi devono accettare di smettere di lavorare
    - Se un programma si pianta mentre processa un messaggio si pianta tutto windows…
    - In windows 3.0 le app windows erano ancora tutte cooperative multitasking (quelle dos ovviamente no, facevano preemptive mt)
  - Windows 3.0 (95/98/Me) VS nt (2k XP)
  - Per saperne di più cercate nell'help "happy anniversary windows"

CWinApp

- In pratica fa partire in winmain e chiama:
- CWinApp.**InitInstance**()
- CWinApp.**Run**()
  - Main loop messaggi
  - CWinApp.**OnIdle**
  - E in risposta ad un messaggio **WM_QUIT** chiamerà CWinApp.**ExitInstance**()
- Queste funzioni di CWinApp sono ovviamente virtual e overridabili (InitInstance **deve** essere overridden)

CWinApp InitInstance

- Scopo inizializzare l'applicazione
- Una delle cose che si può fare a questo livello è creare la finestra principale dell'applicazione
- Nella maggior parte dei casi questo verrà fatto automaticamente dal framework o dai wizard
- Uno dei membri di tale classe è `CWnd* m_pMainWnd;`
- Dovremo creare un oggetto derivato da CWnd e attaccarcelo…

CWnd

- Classe di base per la rappresentazione di generiche finestre
- Da questa classe sono derivate sottoclassi per i più comuni tipi di finestre:
  - Frame Window, Dialog, Views, Buttons

CFrameWnd

- Classe Derivata da CWnd
- Fornisce le funzionalità e i membri per la gestione di una frame window (finestra con barra per il titolo, bottoni di chiusura, minimiz., maxim., bordo per il resize ecc.)
- Di solito si deriva una classe da **CFrameWnd** e vi si aggiunge variabili membro per mantenere dati specifici alla propria applicazione

Hello World con le MFC

- Due classi in gioco, finestra e applicazione
  - Derivare la classe CHelloWnd da CFrameWnd
  - Derivare la classe CHelloApp da CWinApp
- Definire una nuova funzione di inizializzazione per l'applicazione che crei, mostri e faccia aggiornare la finestra.
- Dichiarare un'istanza di un CHelloApp

La minima applicazione

```
#include <afxwin.h>
// Define a window class derived from CFrameWnd
class CHelloWindow : public CFrameWnd {
public:
    CHelloWindow(){
        Create(NULL,"Hello World!",WS_OVERLAPPEDWINDOW, rectDefault); }
};

// Define an application class derived from CWinApp
class CHelloApp : public CWinApp{
public:
    virtual BOOL InitInstance();
};

// Construct the CHelloApp's m_pMainWnd data member
BOOL CHelloApp::InitInstance(){
    m_pMainWnd = new CHelloWindow();
    m_pMainWnd->ShowWindow(m_nCmdShow);
    m_pMainWnd->UpdateWindow();
    return TRUE;
}
// HelloApp's constructor initializes and runs the app
CHelloApp HelloApp;
```


Header precompilati

- Servono essenzialmente per velocizzare la compilazione di header enormi
  - Stdafx.h
    - Contiene tutti gli include di sistema standard e un bel po' di #define utili
  - Stdafx.cpp
    - Necessario per fare gli header precompilati
    - Contiene solo la riga #include "Stdafx.h"

La davvero minima applicazione

```
#include <afxwin.h>
// Define an application class derived from CWinApp
class CHelloApp : public CWinApp{
public:
   virtual BOOL InitInstance();
};

// Construct the CHelloApp's m_pMainWnd data member
BOOL CHelloApp::InitInstance(){
   m_pMainWnd = new CFrameWnd();
   ((CFrameWnd *)m_pMainWnd)->Create(NULL,"Hello
   World!",WS_OVERLAPPEDWINDOW);
   m_pMainWnd->ShowWindow(m_nCmdShow);
   m_pMainWnd->UpdateWindow();
   return TRUE;
}
// HelloApp's constructor initializes and runs the app
CHelloApp HelloApp;
```

Hello World 2

- L'esempio precedente era minimo
- Aggiungiamo:
  - La stringa "Hello World" nel centro della finestra
  - Una finestra di dialogo di about

Hello World 2

- Ancora una volta deriviamo due classi
  - Una per l'applicazione CHelloApp
  - una per la finestra CHelloWindows
- La scritta nel mezzo della finestra deve essere disegnata dall'applicazione:
  - Gestione del messaggio ON_WM_PAINT
  - Ecco perchè ci serviva derivare da cframewnd

Gestione Messaggi

- Filosofia di fondo:
- Il sistema avvisa l'applicazione che è successo qualcosa mandandogli un messaggio
- l'applicazione deve gestire i messaggi
- Le MFC incapsulano il meccanismo di base della gestione dei messaggi di Windows con una serie di *MACRO*

MFC (Visual C++) e Messaggi

- Nella dichiarazione della classe

```
afx_msg void OnPaint();
DECLARE_MESSAGE_MAP()
```

- Nel file dove sono definiti i membri

```
BEGIN_MESSAGE_MAP( CHelloWindow, CFrameWnd )
  ON_WM_PAINT()
END_MESSAGE_MAP()
```

- Tutte queste macro sono gestite bene dall ide del .net

Device Context

- Un device context è una struttura di Windows che contiene informazioni sulle proprietà e attributi di un device (e.g. schermo, stampante)
- Tutte le operazioni di disegno sono fatte attraverso un device context. I Device context permettono di disegnare in maniera device-independent sullo schermo, sulla stampante o su di un metafile.
- Se si vuole opengl occorre gestire anche i rendering context di OpenGL

MFC e Device Context

- Esiste una classe generale delle mfc CDC e varie sottoclassi tipo CPaintDC o CClientDC
  - L'oggetto **CDC** fornisce funzioni membro per lavorare con un device context come la stampante, lo schermo o il display context associato con la client area di una finestra
  - Tutte le operzioni di disegno devono essere fatte attraverso funzioni membro di un CDC

Scrivere in un CDC

- In risposta alla richiesta di ridisegnare la finestra si chiama la OnPaint()

```
void CHelloWindow::OnPaint()
{
   CPaintDC dc(this); // device context for painting
   CRect rect;
   GetClientRect(rect);

   dc.SetTextAlign(TA_BASELINE | TA_CENTER);
   dc.TextOut(rect.right/2,rect.bottom/2,
                                  "Hello World");
}
```


Cambiamo font

- Nei DC ci sono una serie di oggetti correnti tra cui
  - Font
  - Brush
  - Pen
- Sono usati per le varie operazioni di disegno
- Sono oggetti che vanno creati, e distrutti (volendo).


```
CPaintDC dc(this); // device context for painting
CFont font;
VERIFY(font.CreatePointFont(720, "Comic sans MS", &dc));
CRect rect;
GetClientRect(rect);

// Do something with the font just created...
CFont* def_font = dc.SelectObject(&font);
dc.SetTextAlign(TA_BASELINE | TA_CENTER);
dc.TextOut(rect.right/2,rect.bottom/2, "Hello World");
dc.SelectObject(def_font);

// Done with the font. Delete the font object.
font.DeleteObject();
```


Hello World 3

- L'esempio precedente era minimo
- Aggiungiamo:
  - Una finestra di dialogo di about
- Entrano in gioco le ***RISORSE***

Resource Files

- Le *risorse (resource files)* sono i dati (binari o ascii) che un compilatore di risorse o lo sviluppatore aggiungono al file eseguibile di una applicazione
  - Risorse standard descrivono icone, cursori, bitmap, dialog box, fonts, testo di messaggi, ecc.
  - Risorse definite dall'applicazione possono contenere qualsiasi dato richiesto da una specifica applicazione.
  - Ogni risorsa è associata ad un unico numero.

Risorse e Visual C++

- Il visual c++ mette a disposizione tool per la creazione e l'integrazione di risorse in un'applicazione.
- Oltre alla dialog box aggiungiamo le seguenti risorse:
  - menu
  - accellerator

Risorse

- Due file
  - **HelloMfc.rc**
  - Contiene effettivamente un po' dei dati delle varie risorse (dialoghi ecc).
  - **Resource.h**
  - #define che servono per riferire poi le varie risorse
  - Di solito deve essere incluso da tutti i file del progetto
- Tutte gestite in maniera ragionevole dall'ide
- Di solito.
- In realtà dentro resource.h vengono anche nascosti dati dell'IDE…

Dialog Box

- Sono definiti nelle risorse
- Ad ogni dialogo si associa una classe che lo gestisce.
- Per far vedere un dialogo si deve creare un oggetto di quel tipo e poi farlo partire
- Il modo più semplice per invocare un dialogo è quello di farlo modale:
  - L'applicazione che lo ha invocato è fermata (ragionevolment) finchè non si preme ok o cancel.

```
CAboutDlg dlgAbout;
dlgAbout.DoModal();
```


Conclusione

- L'esempio visto oggi è semplicissimo, ma ha il difetto di non rientrare in nessuno degli standard di applicazione mfc (dialog based e document/view)
- Nessun wizard ci fa il setup dell'applicazione da zero
- Alcune delle feature del IDE non funzionano benissimo…
- Le app standard si integrano meglio nell'ide ma sono VERBOSE.