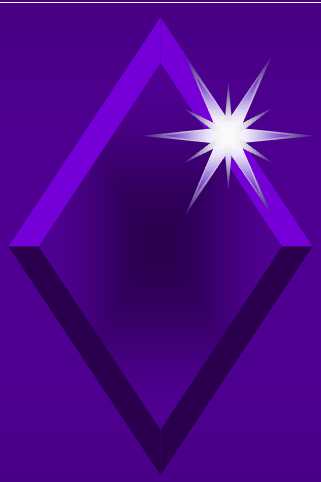
Microsoft Foundation Classes



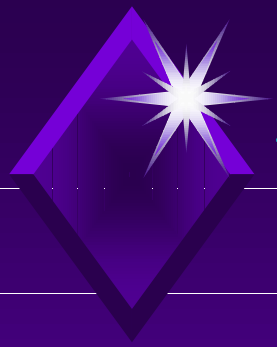
Scott M. Lewandowski

scl@cs.brown.edu

Department of Computer Science

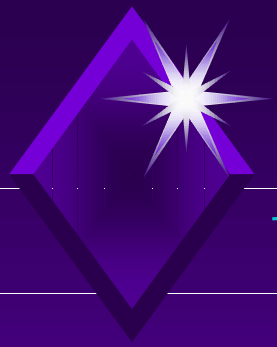
Brown University

Providence, RI 02912

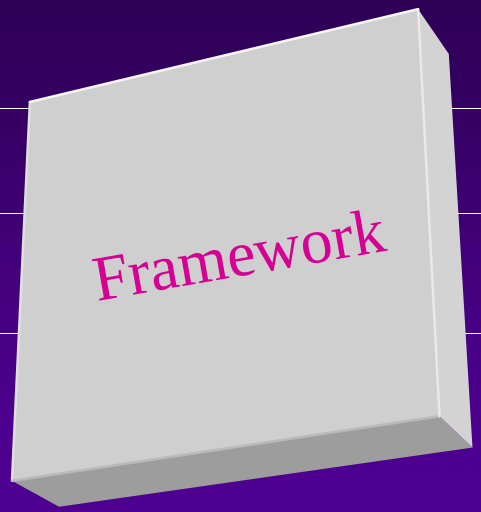


What Is MFC?

- Collection of pre-built components
- Portable interface library
- Scalable application architecture
  - Source is even available!
- Framework supporting expected application functionality



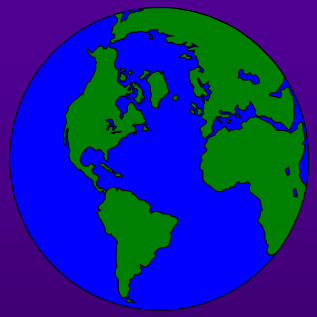
MFC As a Framework



- White Box
  - Extensibility via OO features
  - Requires knowledge of MFC structure
  - Apps bound by framework structure
- Limited programming style options
- Most app design predetermined
- Its products generally work well with it

High-Level Overview

- Two types of classes
  - Windows specific
  - General purpose
- "Base types"
- Asynchronous event model
- Document/view model
- Debugging/Diagnostic support

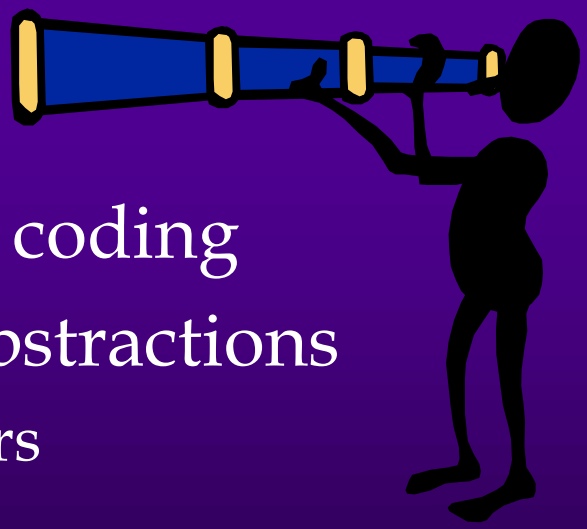


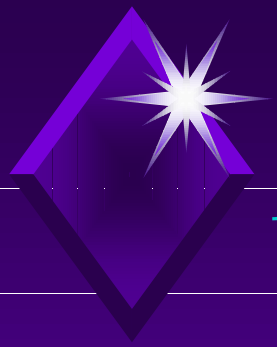
Problems MFC Solves

- Saves development time
- Application portability
  - Via Windows Portability Layer (WPL)
- Provides consistency between apps
- Assured interoperability
  - Between applications (via Windows)
  - Between objects in a system

History of MFC

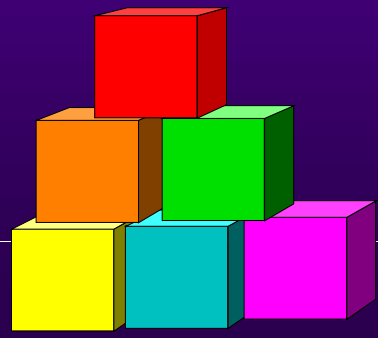
- Continued refinement and evolution
  - Lately has cluttered the interface
  - Now up to 100+ classes
- Change in focus
  - Original goal was to reduce coding
  - Current goal is high-level abstractions
    - Leaves details to programmers
  - Now have customizable classes



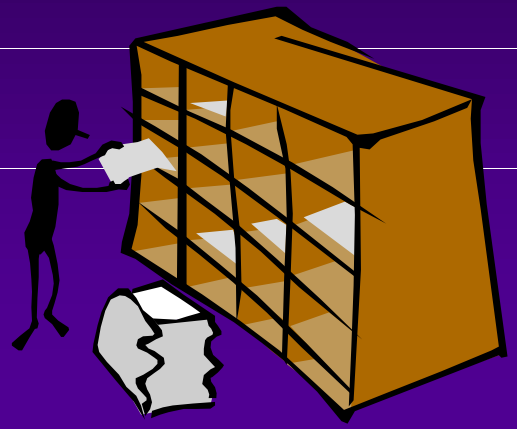


MFC: The Early Years

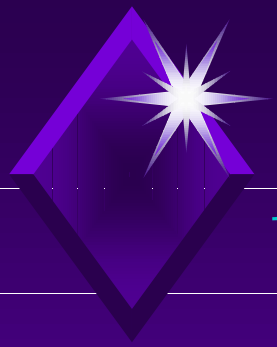
- Pre-MFC: AFX
  - *A*pplication *F*rameworkX
  - Too-removed from Windows APIs
  - Many functions from AFX are now in MFC
- v1.0 (April 1992)
  - Supported general and Windows classes
  - Very thin abstraction
  - Focus was on building blocks



MFC: Important 2.x Releases



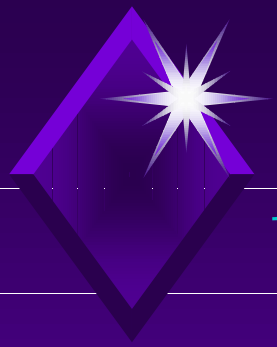
- v2.0 (February 1993)
  - New architecture classes
    - Organize and structure programs
    - Document/View framework debuts here
  - Shift to high-level abstractions
    - Move towards customizable classes
- v2.5 (December 1993)
  - Added database and OLE support
  - Set the stage for non-essential components



MFC: Critical Refinement

- v3.0 (September 1994)
  - First 32-bit only release
  - New user interface idioms
  - Improved language support
  - More stable and complete Win32 support
- v3.1 & v3.2 (January & July 1995)
  - Added Windows Sockets, Win95 controls, swap-tuned DLLs, and new demos

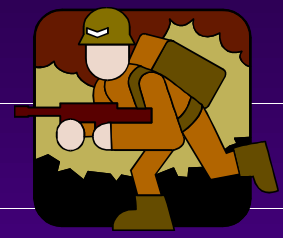
32-bit!



MFC: The Current Generation

- v4.0 (November 1995)
  - Thread synch, OLE controls, DAO added
  - Win95 controls refined
- v4.1 (last time for Win32s support)
- v4.2/v4.2b (ISAPI classes and bug fixes)
- v4.21 (March 1997)
  - Adds IntelliMouse support
  - Numerous bug fixes

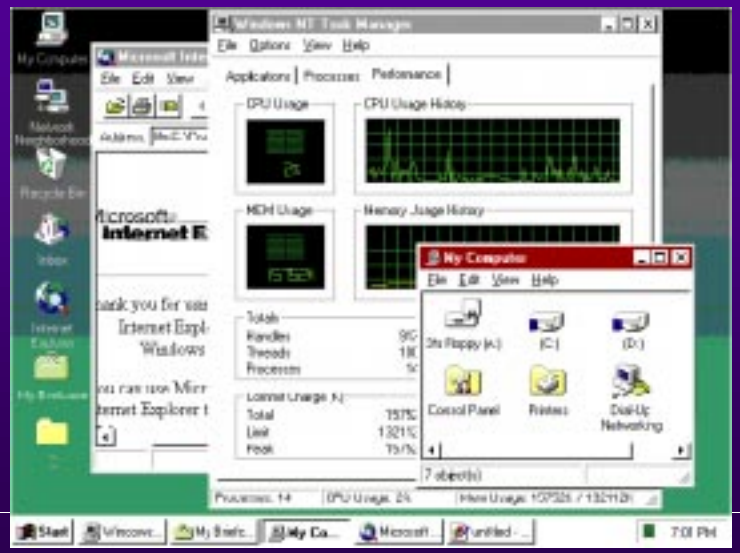
General Purpose Classes

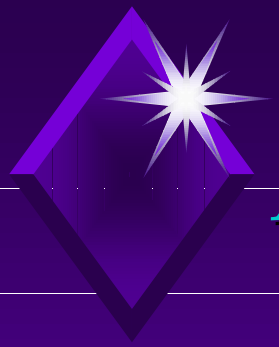


- Handle non-graphical aspects of an app
  - RTTI: for casting
  - Persistence: handles collections and cycles
  - Collections: small variety; not templated
  - Strings: robust, but not char* compatible
  - Files: disk and memory; abstracts clipboard
  - Date and time: hides internal representation

Windows Specific Classes

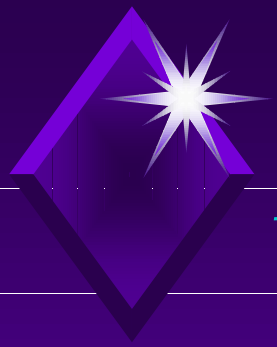
- Manage all graphical aspects of app
- Integrate with Win32
- Basic app components
  - Window management
  - Dialogs
  - Menus
  - OLE





Architecture Class Approach

- Organizes and structures programs
- Support for important application areas
- Needed since no logical and obvious place for some app functionality
- Focus on components critical to many parts of an apps (commands, views, help, documents, etc.)



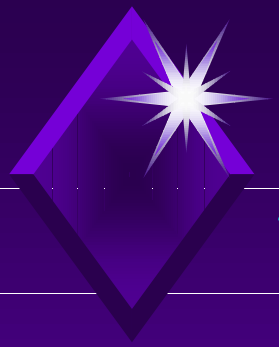
High-Level Abstractions

- Classes not enough
- Encompasses most common UI idioms
- Few lines of code gives you robust functionality (text processing, split scrolling windows, tool bars, etc.)

Not Quite RAD

"Base Types"

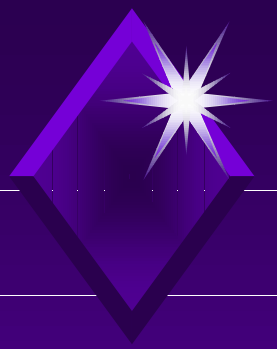
- An attempt to hide machine from the programmer
- Liberal use of typedefs and unions
- Can be very confusing… (a small sampling!)
  - BSTR A 32-bit character pointer
  - DWORD A 32-bit unsigned integer or the address of a segment and its associated offset.
  - LONG A 32-bit signed integer
  - LPARAM A 32-bit value passed as a parameter to a window procedure or callback function
  - LPCSTR A 32-bit pointer to a constant character string
  - LPSTR A 32-bit pointer to a character string
  - LPVOID A 32-bit pointer to an unspecified type
  - UINT A 16-bit unsigned integer on Windows versions 3.0 and 3.1; a 32-bit unsigned integer on Win32
  - WNDPROC A 32-bit pointer to a window procedure
  - WORD A 16-bit unsigned integer



Event Model

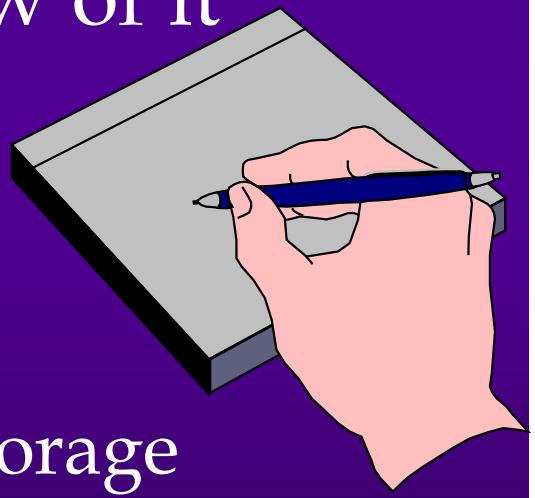
- Directed by the "On…" functions
  - Most of this handled by MFC
- Message/handler approach
- Message categories

Type	Name	Sent By	Handled By
Windows Messages	WM_XXX (not WM_COMMAND)	Windows OS	Windows and Views
Control Notifications	WM_COMMAND, BN_CLICKED	Controls, Client Windows, and Buttons	Parent Windows
Command Messages	WM_COMMAND	UI Objects	Framework hands off to object



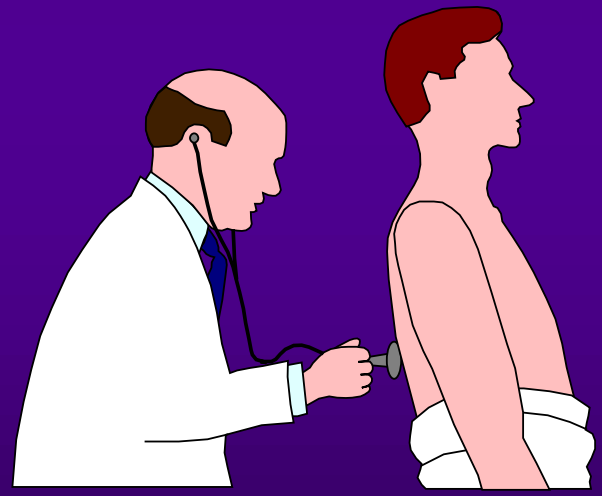
Document/View Architecture

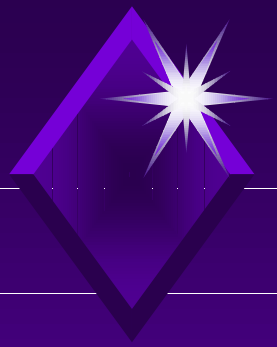
- Derived from Smalltalk's MVC model
- Separates data from user's view of it
  - Separates programming tasks
  - Multiple views possible
- Supported by
  - CDocument: application data; storage
  - CView: "window on data"; display & printing
- Communicate messages updating state



Debugging and Diagnostics

- Debug vs. Release builds
- Tracks all objects allocated on heap
- Validates pointers
- Robust ASSERT macro
- Two debug output macros
  - TRACE (C style printf)
  - afxDump (C++ style streams)
- State verification





Structured Exception Handling

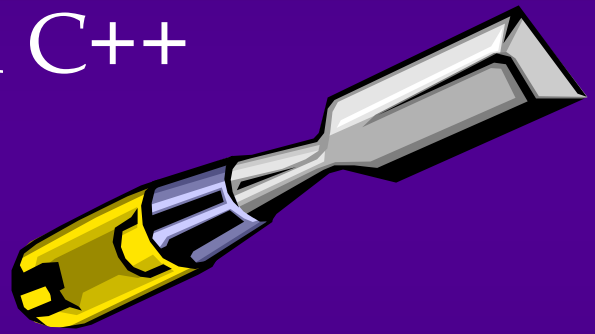
- Complements C++ exception handling
- Gain control of app when it would normally terminate
- Kernel supported (good for threads)

```
__try {
        // some code here
} __ finally {
        // completion code (always executed)
        if(AbnormalTermination()) {
                // error code
        }
}
```

```
__try {
        // some code here
} __ except(filter) {
        // exception handler (executed if filter returns)
        // EXCEPTION_EXECUTE_HANDLER
}
```

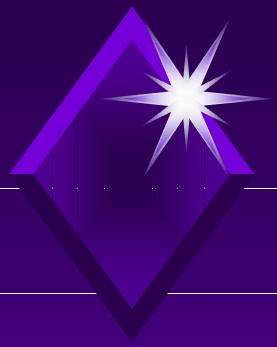
Integration With Tools

- Use of tools almost essential
- Some examples from Visual C++
  - Class Wizard
  - App Wizard
  - Component Gallery
- Simplifies development
- Reduces programming errors



Practical Issues

- Ease of use/learning curve
  - Automated tools help *and* obfuscate
  - Inconsistent and unintuitive interfaces
  - Microsoft: 6-12 months to be effective
  - Writing an app vs. writing it correctly
- Lots of quirks and bugs
  - Poor testing
  - Even errors in simple math functions



Some Microsoft Weirdness…

- C++ functionality tied to Win32
  - Supposedly a strength
- "Advanced issues"
  - Copy constructors, assignment operators, correct object destruction
- Reliance on macros
- Incompatibly between Afx, MFC, Win32
- Environment generated code (not seen)

Microsoft®

Questions