User Interfaces

- Connection between the computer and the user
- Two types:
  - Command Line
  - GUI—Graphical (Visual)

Command Line Interfaces

- User types commands ==> must remember
- Results Scroll by
- Text-based
- “Interactive” but hard to use
- Flow of info: keyboard --> program--> Display
- No direct interaction between user and screen

Visual (Graphical) Interfaces

- Show Graphical Objects (images, icons, buttons, scroll bars) on screen
- User interacts using pointing device
  - Direct, intuitive, intimate interaction
- Objects can be dragged, buttons pushed, etc....
- Better way of using screen space
  - Panes can overlap
  - Underlying panes can be brought to forefront
  - Desktop metaphor (like papers on a desk)
    - Well, not exactly!

Graphical Interfaces, Continued

- Use graphics to organize user workspace
- Present user intuitive ways of accomplishing tasks
  - e.g., copy files by dragging
- Environment allows many tasks to be performed simultaneously
  - Different tasks share screen space
- Visually rich way of conveying information
- WYSIWYG display of documents
Main Feature of GUIs:

THE WINDOW
- Rectangular area of screen onto which a program draws text and graphics.
- User interacts with program using pointer device to select objects inside.
- Some window components:
  - border, title bar, client area, menu bar, scroll bars, max/min/close buttons, tool bars, etc.

Brief History of GUIs
- 1968, ARPA-funded Stanford Research Center (Doug Englebart)
- First windows (screen sliced up into overlapping panes)
- Only textual info
- Underlying windows can be popped to the top
- Selection done with light pen
- Invented the mouse

Xerox PARC--Alto Computer
- 1970
- First GUI
- Cursor tracked position of mouse
- WYSIWYG
- Windows with precise text
- Displayed more than just text
- First interactive painting program

Recent History (PCs)
- 1983, Apple Lisa (failure)
- 1984 Apple Macintosh--standard for GUIs
- 1985 Microsoft releases Windows 1.0
  - Difficult to program
  - Prone to crashing
  - Needed hardware not yet available
- 1987 Windows 2.0 (still real mode only)
- 1988 Windows/386 (Virtual 86 mode on 386==>multiple DOS sessions in windows)

Recent History (Microsoft)
- 1990 Windows 3.0
  - 80x86 protected mode, up to 16Meg memory, cooperative multitasking
  - TrueType fonts, multimedia, protected mode only; Networking
- 1993 Windows NT
  - 32-bit flat memory space, 16 MB, thread-based preemptive multitasking, separate from DOS, multi-platform, networking, secure
- 1995 Windows 95
  - Runs on 4 Meg, long file names, plug and play, new controls, new desktop/window style
  - Hybrid 16/32 bit OS, depends on DOS, lacks security of NT, no portability to RISC
- 1998 Windows 98
  - Web-like interface, legal issues
Other GUI-Windowing Systems
- IBM OS/2: Presentation Manager
- Commodore Amiga: Intuition
- Atari: GEM
- Sun Microsystems: NeWS
- The X Window System
  - Developed at MIT, networked graphics programming interface, independent of machine architecture/OS (but most used under UNIX)

Workshop Content
- Microsoft Windows Visual C++
  - Using Microsoft Developer Studio (Visual C++ Development Environment)
  - Win32 API Programming
  - MFC Programming
  - Syllabus, Example programs and notes online at:
    - http://www.cs.binghamton.edu/~reckert/
    - “Visual C++ Programming Workshop” link

Win32 API Programming
- Event-Driven Programming (Messages)
- Menus and other Resources
- Text and Graphics
- Mouse and Keyboard
- Bitmaps, Animation, Timers
- Child Window Controls
- Child and Popup Windows
- Dialog Boxes
- The Clipboard

Microsoft Foundation Class (MFC) Programming
- The MFC Class Hierarchy
- The Application/Window Approach
- The Document/View Approach
- Using “AppWizard” & “ClassWizard”
- Drawing, Menus, & Dialog Boxes with MFC
- File Handling and Printing
- Dialog-Based MFC Applications and Common Dialog Boxes
- Windows Multimedia
- Network Programming (TCP/IP) with MFC
- HTML-based Applications with MFC

Features of Windowing Systems
- Consistent user interface
- Display within a window
- Menus to initiate program functions
- Make use of controls:
  - predefined windows used with main program window
  - examples: buttons, scroll bars, edit controls, list boxes, drop-down list boxes
  - Dialog box--popup window containing several controls

Consistent User Interface
- Programs have same look and feel
- Same built-in logic to:
  - draw text/graphics
  - display menus
  - receive user input
  - controls, dialog boxes, use of mouse
Multitasking
- Every program acts like a RAM-resident popup
- Programs run “simultaneously”
- Each program’s output occupies its own window
- Windows can be moved and sized
- User can switch between programs

Windows Multitasking Features
- Cooperative (Windows 3.xx)
  - Programs must give up control so others can run
  - Programs coexist with other programs
- Preemptive (Windows NT, 95, 98)
  - Thread-based–System timer allocates time slices to running program threads
  - Under both systems, code is moved or swapped into and out of memory as needed

Windows Object Orientation
- A window is handled like a C++ object
  - Has a user-defined type (Windows class)
  - Instances of class created at runtime
  - Messages sent to windows affect their behavior

Windows Memory Management
- Older versions: 16-bit, segmented memory
  - Dictated by processor architecture
  - Hard to program
- Newer versions: 32-bit, flat memory model
  - Easier to program
  - As old programs terminate, new ones start; code swapped into and out of memory
  - Fragmentation can occur
  - Windows must consolidate memory space
  - Moves blocks of code/data continually

Memory Management, continued
- Several instances of a program
  - code only loaded into memory once
  - program instances share same code
- Programs can share code located in other files (Dynamic linking)

Static vs. Dynamic Linking
- Static Linking
  - code incorporated into executable at link time
- Dynamic Linking
  - Linker generates relocation info
  - Put into executable
  - DLL loaded when needed
  - Relocation info used to get DLL function code as needed
Pros/Cons of Dynamic Linking

- Smaller programs (code is not there)
- DLL can be used by many programs with no memory penalty
  - Only loaded once!
- Updates to DLLs don’t require recompilation of programs using them
- Disadvantage--DLL must be present at run time==>no standalone programs

Device Independent Graphics Interface

- Windows programs don’t access hardware devices directly
- Make calls to generic functions within the Windows ‘Graphics Device Interface’ (GDI)
- The GDI translates these into HW commands

Device Independent Graphics Interface

- May use device drivers (HW control programs)
- Thus graphics I/O done in a “standard” way
- Programs will run unaltered on other HW platforms

Windows API

- The interface between an application and Windows
- A library of functions Windows programs can call
- Several versions
  - Win16 (16 bit apps for Windows 3.xx)
  - Win32 (32 bit apps for Windows NT/95)
  - Win32s (patches Win16 to create 32 bit apps that run under Windows 3.xx)
Classical Windows programming
- Use C to access raw API functions directly
- No C++ class library wrappers to hide API
- Hard way to go, but most basic & flexible
- Provides understanding of how Windows and application program interact
- Establishes a firm foundation for MFC programming
- We will try to do both

Class-based Windows programming
- Microsoft’s MFC Library
- Borland’s OWL Library
- Encapsulate the API functions into classes
- Provide a logical framework for building windows applications

MFC Library
- Microsoft’s C++ Interface to Windows API
- O-O Approach to Windows Programming
- Some 200 classes
- API functions encapsulated in the MFC
- Classes derived from MFC do grunt work
- Just add data/functions to customize app
- Provides a uniform application framework

Microsoft Visual C++
- 2 Windows app development systems
  - C programs using Win32 API
  - C++ programs using MFC
- Some Developer Studio IDE Components
  - Text/Resource Editors
  - C/C++, Resource Compilers
  - Linker
  - Debugger
  - Wizards
  - On-line Help

Some MFC Characteristics
- Reusable code
- Smaller executables
- Faster program development
  - But a steep learning curve is required
  - And there is less flexibility
- Programs must be written in C++
- Require the use of classes==>
  - Programmer must know OOP
Sequential Programming
- Standard programming—program solicits input (polling loop)
- Approach follows a structured sequence of events
- Example—averaging grades:
  - Input name
  - Input first grade
  - Input second grade
  - Input third grade
  - Calculate average
  - Output average

Event-Driven Programming
- Designed to avoid limitations of sequential, procedure-driven methodologies
- Process events as they happen—non-sequential
- Program doesn’t solicit input
- OS detects an event has happened (e.g., there’s input) and sends a message to the program
- Program then acts on the message
- Messages can occur in any order

Sequential vs. Event-Driven Programming
- Standard Sequential programming:
  - Program does something & user responds
  - Program controls user (the tail wags the dog)
- Event-Driven Programming:
  - Used by Windows
  - User can act at any time
  - User controls program (the dog wags the tail)
  - OS really is in control (coordinates message flow to different applications)
  - Good for apps with lots of user intervention