CS-360
GUI & Windows Programming
Dr. Richard R. Eckert
Computer Science Department
SUNY Binghamton
Fall, 2004
MWF, 10:50-11:50 A.M.
S2-337

Course Information
Office: EB-N6
Phone: 777-4365
Office Hours: TBA
Email: reckert@binghamton.edu
http://www.cs.binghamton.edu/~reckert/
CS-360 link for syllabus, notes, programs, assignments, etc.
Class Listserv:
cs360-L@listserv.binghamton.edu
TA Information: TBA

Course Prerequisites
- CS-220, Computer Organization and Assembling Language Programming
- CS-240, Data Structures
- Some knowledge of C or C++ helpful
- Not essential

Text Book Information
- Required:
- Recommended:
- Many Books on Reserve

Evaluation
- Programming Assignments 45%
- Term Examinations (2) 40%
- Final Project 15%

Policies
- Assignments
  Individual
  Due on due date, but can be turned in to CS-360 drop drawer outside CS Department any time that day or night
  5% off for every day late
  Weekends and holidays not included
  No assignments accepted more than one week late
- Originality
  Any work found to be copied will be grounds for an F in the course
Course Schedule (weekly)
1. Intro to GUIs & Windows Programming
2. Using Visual Studio, Win32 API Programming
3. MFC Programming: App/Window & Doc/View Approaches
4. Visual Studio .NET & C#, Windows Forms, Events, Essential Structures
5. Graphics, Animation, Timers, DateTime
6. Mouse, Images, Bitmaps
7. Text, Fonts, Keyboard, Printing
8. Pages & Transformations, Menus

Course Schedule (continued)
9. Controls: Buttons, Labels, TestBoxes, Scrollbars, Listboxes
10. Dialog Boxes, Common Dialog Boxes, File/Stream I/O
11. Clipboard, Multimedia
12. Network Programming, TCP/IP Sockets
13. Data Bases and ADO.NET, Web Matrix
14. XML, Web Forms, Web Controls, ASP.NET
15. ASP.NET Web Services
16. The X Window System

Introduction To GUIs and Windows Programming

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
- No direct interaction between user and screen

Visual (Graphical) Interfaces
- Show Graphical Objects on screen
  - e.g., images, icons, buttons, scroll bars
- User interacts using pointing device
- Intuitive
  - 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
- 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, tool bars, scroll bars, max/min/close buttons, etc.

Brief History of GUIs

- 1968: ARPA-funded Stanford Research Center (Doug Engelbart)
- First windows (screen sliced up into overlapping panes)
- Only textual information
- Underlying windows could be popped to the top
- Selection done with light pen
- Invented the mouse

Xerox PARC--Alto Computer

- 1970s
- First GUI
- Cursor tracked position of mouse
- WYSIWYG
- Windows with precise text
- Displayed more than just text
- First interactive painting program
- Technology “acquired” by Apple

Recent History (PCs)

- 1977: Radio Shack TRS-80, Commodore Pet, Apple II
- 1981: IBM PC, DOS
- 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
- 1988: Windows/386 (Virtual 86 mode on 386==x*multiple DOS sessions in windows)

Recent History (Microsoft)

- 1990: Windows 3.0
  - 80x86 protected mode, up to 16 Meg memory, cooperative multitasking
  - TrueType fonts, multimedia, protected mode only: Networking
- 1993: Windows NT
  - 32-bit flat memory space, 16 MB, thread-based pre-emptive multitasking, separate from DOS, multi-platform, networking, secure
Recent History (Microsoft)
- 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
  - Integrated Web functionality
  - Upgrades of 95-98-NT
  - 95->98->Me->XP Home: for home use
  - NT->2000->XP Professional: for businesses
  - XP:
    - fancier user interface; latest multimedia (DVD); upgraded web capabilities; improved help (remote); improved performance & security

Recent History (Microsoft)
- 2000: The Microsoft .NET Initiative
  - A new paradigm for Windows distributed applications
  - Independence from specific language or platform
  - Applications developed in any .NET compatible language
    - Visual Basic .NET, Visual C++ .NET, C# and more
  - Programmers can contribute to applications using the language in which they are most competent
  - Architecture capable of existing on multiple platforms
  - New program development process
    - Provides increased productivity
    - Vision for embracing the Internet in software development
    - New way of designing & creating applications that share work between components (local and distributed over the internet)

Other GUI-Windowing Systems
- IBM OS/2: Presentation Manager
- Sun Microsystems: Java
  - AWT
  - Swing
  - Platform independent
  - JDK is free
- The X Window System
  - Developed at MIT, late 1980s
  - Networked graphics programming interface
  - Independent of machine architecture/OS (but most used under UNIX)

Course Content
- Microsoft Windows Visual Studio .NET
  - Using Microsoft Developer Studio (Visual Studio .NET)
  - Win32 API Programming and MFC Programming using Visual C++
  - The .NET Framework: Programming Windows Forms, Web Applications, Web Services, and Data Base Applications using C#;
  - Introduction to X-Windows Programming
  - Example programs and notes online at:
    - http://www.cs.binghamton.edu/~reckert/
    - "CS-360" link

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

Consistent User Interface, continued
- 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 occupies its own window
- User interacts with program in its window
- User can switch between programs

Windows Multitasking Features

- Cooperative (Windows 3.xx)
  - Programs give up control so others can run
  - Programs coexist with other programs
- Preemptive (Windows NT, 95, 98, XP)
  - 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 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
- Windows does this automatically
- 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
  - Code is put into separate modules
    - These are loaded at run time
  - Linker generates relocation information
    - Only that is put into executable
    - Smaller programs
  - DLL loaded when needed
  - Relocation info used to get DLL function code as needed

Static Linking

![Static Linking Diagram](image1)

Dynamic Linking

![Dynamic Linking Diagram](image2)
Pros/Cons of Dynamic Linking

- Smaller programs (code is not in program)
- DLL can be used by many programs with no memory penalty
- Only loaded once!
- Disadvantage--DLL must be present at run time ==> no standalone programs
- Most of the Windows OS is implemented in DLLs

Device Independent Graphics

- 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

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/XP)
  - Win32s (patches Win16 to create 32 bit apps that run under Windows 3.xx)

Classical Win32 API 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
- Faster executables
- Provides understanding of how Windows and application program interact
- Establishes a firm foundation for MFC and .NET programming

Class-based Windows Programming

- Microsoft Foundation Class Library (MFC)
- Microsoft .NET Framework Class Library (FCL)
- Borland’s OWL Library
- Characteristics:
  - Encapsulate the API functions into classes
  - Provide a logical framework for building Windows applications
  - Object Orientation means reusable code

MFC Library

- Microsoft’s C++ Interface to Win32 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 .NET

- What is it?
  - A platform to run code on
  - A class library of code that can be used from any language (FCL)
  - New programming interactive development environment
  - A set of server products
  - New way of designing & creating applications that share work between components (local and distributed over the internet)

- You can get it free from the Watson School Microsoft Academic Alliance
- It’s huge!

Components of .NET

- The .NET Framework Class Library (FCL)
  - Organized into namespaces (like packages in Java)
  - Handle things like: Data, IO (simple & file), Windows Forms, Web Forms, Windows Controls, User Interfaces, Drawing, Threading, Exceptions, Networking, Web Services, Data Bases (ADO), XML, ASP, Security, Collections, … lots of others
- Common Type System (CTS)
- Common Language Specification (CLS)
- Common Language Runtime (CLR)

Microsoft Visual Studio

- Developer Studio IDE (Interactive Designer)
- 3 Windows application development systems
  - C/C++ programs using Win32 API
  - C++ programs using MFC
  - Multilanguage program development using .NET Framework Class Library & the CLR
- Some Developer Studio IDE Components
  - Text/Resource Editors
  - C, C++, C#, Visual Basic, J#, etc. Language Compilers
  - Resource Compilers
  - Linker
  - Debugger
  - Wizards
  - On-line Help

NET Framework

- Platform for developing distributed applications for the Internet
- Design Goals:
  - Provide high degree of language interoperability
  - Provide a managed runtime environment
  - Provide simple software deployment & versioning
  - Provide high-level code security through code access security & strong type checking
  - Provide consistent object-oriented programming model
  - Facilitate application communication by using industry standards such as SOAP & XML
  - Simplify Web application development with ASP.NET
  - Facilitate Data Base access with ADO.NET
  - Provide high performance and easy scalability

NET Architecture
Compilation in the .NET Framework

.NET Framework and the Common Language Runtime

- Why two compilations?
  - Platform independence
    - .NET Framework can be installed on different platforms
  - Execute .NET programs without any modifications to code
- Language independence
  - .NET programs not tied to particular language
  - Programs may consist of several .NET-compliant languages
  - Old and new components can be integrated
- Other advantages of CLR
  - Execution-management features
    - Manages memory, garbage and other features
    - Relieves programmer of many responsibilities
    - More concentration on program logic

Sequential Programming (Console Applications)

- 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, etc.
  - Calculate average
  - Output average

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 does something and program responds
  - 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

Event-Driven Programming

- Designed to avoid limitations of sequential, procedure-driven methodologies
- Process user actions (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

The Event-Driven Programming Paradigm