CS-360
GUI & Windows Programming

Dr. Richard R. Eckert
Computer Science Department
SUNY Binghamton
Fall, 2005

Section 01: MWF, 8:30-9:30 A.M.
Section 02: MWF, 1:10-2:10 P.M.
SL-302

Course Information

- Office: EB-N6
- Phone: 777-4365
- Office Hours:
  - T 10:00-11:30 A.M.
  - W 2:30-3:30 P.M.
- Email: reckert @binghamton.edu
- http://www.cs.binghamton.edu/~reckert /
  - CS-360 link for syllabus, notes, programs, assignments, etc.
- Class Listservs:
  - CS360-01@listserv.binghamton.edu
  - CS360-02@listserv.binghamton.edu
- TA Information: TBA

Course Prerequisites

- CS-240, Data Structures
- Knowledge of C or C++

Text Book Information

- Required:
  - Deitel, et.al., "C# for Experienced Programmers", PH/Pearson, 2003, ISBN 0-13-046133-4
- Recommended:
  - Kate Gregory, "Special Edition Visual C++ 6 .NET", Que, 2002, ISBN 0-7887-2466-9
  - Bradley & Millspaugh, "Programming in C# .NET", McGraw-Hill, 2004, ISBN 0-07-121564-6
- Many Books on Reserve
  - See Reserve List on Course Syllabus

Software

- Microsoft Visual Studio .NET (2003)
  - Available at University public computer facilities
  - Get your own copy
    - From Microsoft Academic Alliance
      - All registered BU students
      - Will be made accessible after "Add/Drop" deadline
    - Also comes with Bradley & Millspaugh book
- Microsoft ASP.NET Web Matrix
  - Available at University public computer facilities
  - Free from Microsoft
    - asp.net/webmatrix
- Microsoft Visual Web Developer Express (Beta)
  - Free from Microsoft

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, TextBoxes, 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: DARPA-funded Stanford Research Institute (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
  - Wozniak & Jobs … Gates
    - Book: "Fire in the Valley"
    - Movie: "Pirates of Silicon Valley"

Recent History (PCs)

- 1977: First "Personal Microcomputers:
  - Radio Shack TRS-80, Commodore Pet, Apple II
- 1981: IBM PC, DOS
- 1983: Apple Lisa (failure)
- 1984: Apple Macintosh--standard for GUIs
- 1985: Microsoft released 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==>multiple DOS sessions in windows)

Recent History (Microsoft)

- 1990: Windows 3.0
  - 80x86 protected mode, up to 16 Meg memory, cooperative multitasking
- 1992: Windows 3.1, Windows for Workgroups 3.11
  - 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
  - Ran on 4 Meg, long file names, plug and play, new controls, new desktop/window style
  - Hybrid 16/32 bit OS, depended on DOS, lacked security of NT
- 1998: Windows 98
  - Integrated Web functionality
- 2000-present: Windows 2000, ME, XP, 2003 Server
  - Upgrades of 95-98-NT
    - 95 → 98 → ME →XP Home: for home use
    - NT → 2000 → XP Pro → 2003 Server: for businesses
  - XP, 2003 Server:
    - 64-bit capability; Fancier user interface; latest multimedia (DVD); upgraded web capabilities; improved File/directory services, improved help (remote); improved performance & security; Microsoft .NET Framework SDK built into 2003 Server.

The Microsoft .NET Initiative (2000)

- Requirements for 21st Century Software Development:
  - Internet, WWW, and Intranet - awareness
  - Universal representation of:
    - software models (UML)
    - and data (XML)
  - Enhanced security and reliability
  - *Integration* of various programming languages and tools, databases, knowledge bases and networking tools into a *single infrastructure*
  - Design and development of reusable software components

Microsoft .NET Framework (2000)

- Only software system that addresses new requirements
  - Windows Forms for standalone Windows applications
  - New paradigm for Windows distributed applications
    - ASP.NET – web applications and services
    - ADO.NET – data base integration
  - Independent of language
    - Applications developed in any .NET compatible language
      - Visual Basic, Visual C++, C#, J#, many more
    - Programmers use language in which they are most competent
  - Architecture can exist on multiple platforms
  - New program development process
    - Provides increased productivity
    - Vision for using the Internet in software development
  - New way of designing & creating applications that share work between components (local and distributed over the internet)
  - New security and reliability features

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":
- 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 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

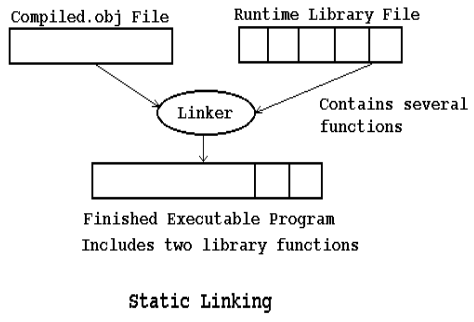
- Cooperative (Windows 3.xx)
  - Programs must give up control so others can run
  - Programs coexist with other programs
- Preemptive (Windows NT, 95, 98, XP, 2000)
  - Thread-based: System timer allocates time slices to running program threads
- Under both systems, code is moved or swapped into and out of memory as needed

Windows Memory Management

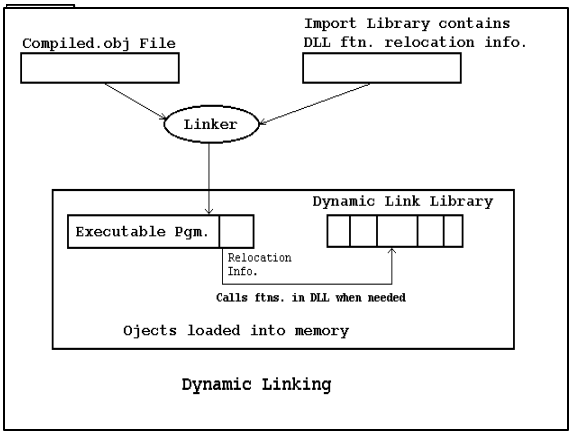
- Older versions: 16-bit, segmented memory
  - Dictated by processor architecture
  - Hard to program
- Newer versions: 32-bit, flat memory model
  - Easier to program
- As old programs terminate, new ones start
  - Code swapped into and out of memory
  - Windows OS does this automatically
- Programs 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 (DLLs)
    - These are loaded at run time as needed
  - 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



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!
- Disadvantages:
  - DLL must be present at run time ==> no standalone programs
  - "DLL Hell" when new versions come out
- 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

Program ⇒ GDI ⇒ Hardware

Windows API

- Application Program Interface
- The interface between an application and Windows
- A library of functions Windows programs can call
- Several versions
  - Win32 API most important
    - (32 bit apps for Windows NT/95/98/XP/2000/2003)

Classical Win32 API Windows programming

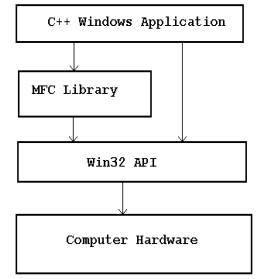
- Use C to access raw API functions directly
- No C++ class library wrappers to hide API
  - But C++ compiler can be used
- Hard way to go, but most basic
- Faster executables
- Provides understanding of how Windows and application program interact
- Establishes a firm foundation for MFC and .NET programming

Class-based Windows Programming

- "Microsoft Foundation Class" Library (MFC)
- Microsoft .NET "Framework Class Library" (FCL)
- Borland's "Object Window Library" (OWL)
- 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
  - Or derive your own classes from MFC classes
- Provides a uniform application framework



The Relationship between Windows MFC and Win32 API Programming

Microsoft Visual Studio

- Developer Studio IDE
- 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
- Details later

Microsoft .NET Framework

- What is it?
  - A platform to run code on
  - Multiple language compilers
  - Common Language Runtime (CLR)
    - Compiler, execution management, much more
  - A class library of code that can be used from any language (FCL)
  - Must be installed to run .NET applications
- You can get it and Visual Studio free from the Watson School Microsoft Academic Alliance
  - It's huge!

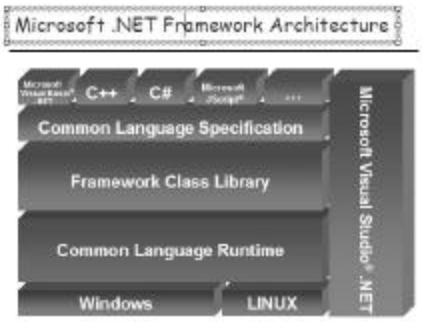
.NET Framework

- 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 development with ASP .NET
  - Facilitate Data Base access with ADO .NET
  - Provide high performance and easy scalability

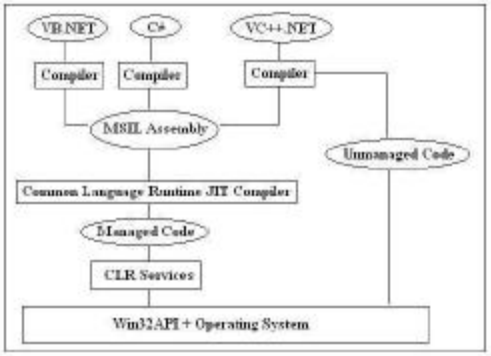
Components of .NET Framework

- Language compilers
- 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)

.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
    - E.g., "Mono": A .NET Development System Project for LINUX
    - Microsoft SSCLI ("Rotor") free academic version of .NET
  - Language independence
    - .NET programs not tied to a 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, security and other features
      - Relieves programmer of many responsibilities
      - More concentration on program logic

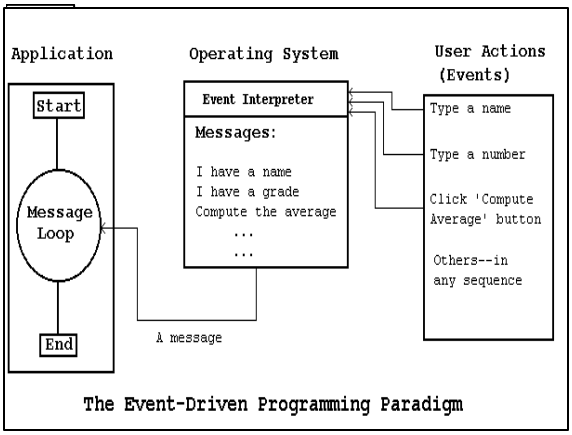
Sequential Programming versus Event-driven Programming

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

Event-Driven Programming

- Designed to avoid limitations of sequential, procedure-driven methodologies
- OS processes 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

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