Document 2003-4

A WIN32 CONSOLE CLASS LIBRARY

Jeffrey S. Franzone, Assistant Professor
Engineering Technology Department
University of Memphis

Abstract

The Console Class Library, Version 2 (CCL2) is an easy to use C++ class that provides many useful routines to increase the functionality and embellishment of Win32 console-mode applications. Although CCL2 was designed primarily as a teaching tool for beginning C++ programmers, it is robust and complete enough to be useful for console-mode industrial-type applications. CCL2 was written and tested with Microsoft Visual C++ 6.0/7.0 and is intended to provide console-mode support for Win32 console-mode applications developed in Microsoft Visual C++ 6.0/7.0 and run on Windows 98/XP.

CCL2 is currently used in the C++ programming curriculum in the Computer Engineering Technology Department at the University of Memphis. The class library is used to introduce students to multiple source files, basic object-oriented terminology and concepts, and ready-to-use class methods that improve and embellish Win32 console-mode application development. Because CCL2 can *"supercharge"* console-mode applications, it is a good choice for those instructors who wish to teach basic C/C++ programming concepts and techniques within a console-mode application environment rather than a Windows application environment.

CCL2 is a great teaching tool because it acts as a bridge between traditional non-GUI programming techniques and more advanced GUI programming techniques that most students will eventually encounter in advanced programming courses. With this library, students can quickly incorporate color, mouse input, and graphics boxes and menus into their console-mode applications while slowly being introduced to basic object-oriented terminology and design. The simplicity of the library (it does not incorporate overloaded operators, inheritance, or polymorphism) makes it a wonderful teaching tool for beginning C++ students as they learn about classes, constructors, destructors, member functions, default arguments, public and private access specifiers, static data members, namespaces, and common Win32 Console API functions all within the context of a *"real-world"* C++ class library.

I. Justification for CCL2

The engineering technology department at the University of Memphis has chosen to emphasize console-mode programming rather than Windows programming for the lower-level programming courses. It is felt that console-mode programming is an easier environment to teach, analyze, and learn fundamental and standard C/C++ programming concepts and techniques. Many of these concepts and techniques, when taught in a console-mode environment, are also more portable to other computer platforms. In the upper-division programming courses, more emphasis is placed on windowing and GUI concepts and techniques using Java rather than C++ in a Windows application environment. Again, it is felt that Java's portability, rather than Windows programming techniques, lends itself better to teaching such user interface techniques.

Many beginning and intermediate C/C++ textbooks explain and demonstrate programming concepts within a *"console-mode application"* environment rather than a *"Windows application"* environment. Listed below are several reasons for this:

1. Console-mode applications are associated with a single, low-level *"console"* (standard input, output, and standard error devices) and a sequential-model of execution that can be easily duplicated on other computer systems. Emphasis is placed on programming techniques and features that are portable and easy to implement on other systems. In contrast, Windows applications and Windows programming techniques are more closely tied to the Windows operating system (which has an event-driven model of execution) and are not generally portable or easily duplicated on other systems.

2. Many fundamental programming concepts (both structured and object-oriented) can best be understood (and taught) within a console-mode environment because of the simplicity and portability of the sequential-model of execution, rather than the event-driven model of execution.

3. In general, Windows applications require more effort to program than console-mode applications because of the complexity of graphics features (such as windows, buttons, and menus) and advanced operating system features such as multitasking and multithreading operations. To help manage the complexity of writing Windows programs, Windows programmers must spend considerable time learning the interface classes and Windows application protocols that are needed to access, manage, and use operating system features and resources and to ensure robust behavior when the application is run with other Windows applications. This is in addition to writing the specific C++ code to make the program perform the desired functions.

4. Attempting to learn fundamental C++ programming concepts within the context of writing Windows applications can be extremely difficult and overwhelming. The task of learning Windows programming in addition to C++ programming is a distraction that can obscure C++ programming comprehension.

II. Introduction to CCL2's Features and Limitations

CCL2 is designed to be extremely easy to use for beginning C++ students. CCL2 consists of two files, "consoleClass.cpp" and "consoleClass.h". To use the class library, students simply add the two files to an existing Visual C++ project, declare an object of class Console in the main source file, and simply use the member functions available. Both files are fairly well-documented as far as descriptions of member functions, side-effects, and the types of arguments required and return values.

With the exception of the class constructor and destructor, CCL2 member functions are divided into three categories: *General-purpose functions, Console Graphics functions, and Extended Keyboard and Mouse functions*. Listed below is a short summary and brief description of the member functions available in CCL2.

Console()
: Default constructor. Gets the handles to the console screen and console input buffer, initializes the internal class handle variables used by most of the console API methods, sets the screen-buffer size, and provides common initialization settings that ensure the screen buffer starts in a consistent state. (NOTE: Screen-buffer sizes less than 80x25 are not supported. Passing in values smaller than this will default to 80x25.)

~Console()
: Default destructor. Closes the handles to the console screen and console input buffers.

General-purpose functions:

clrscr()
: Clears a WIN32 Console mode screen-buffer. By default, an 80x25 console-mode screen buffer is cleared. However, screen-buffer sizes larger than this are automatically cleared. (Smaller screen-buffer sizes are not supported.)

clrline()
: Clears a line or group of lines in a WIN32 Console-mode screen.

sleep()
: Pauses for a specified number of milliseconds. Independent of microprocessor speed.

hideCursor()
: Controls the visibility of the screen cursor in WIN32 Console-mode.

setCursorPos()
: Positions the cursor at a screen location in a WIN32 Console-mode window.

getCursorPosX()
: Fetches the current X screen coordinate of the cursor. (Not valid with standard C/C++ I/O functions.)

getCursorPosY()

Fetches the current Y screen coordinate of the cursor. (Not valid with standard C/C++ I/O functions.)

setConsoleTitle()

Displays a text message on a console window's title bar. This function simply provides a wrapper for the WIN32 console function SetConsoleTitle() so that it can be a member of the Console class.

flushConsoleInputBuffer()

Flushes (or empties) the console input buffer. This function simply provides a wrapper for the WIN32 console function FlushConsoleInputBuffer() so that it can be a member of the Console class.

wait()

Pauses the display indefinitely until a standard key is pressed. (No echo to the screen, removes the character from the input buffer.)

keyhit()

Queries the keyboard buffer for a standard keystroke. (No echo to the screen, removes the character from the input buffer.)

keyMousePress()

Queries the console input buffer for a keystroke or mouse-click and flushes the input buffer. Keystrokes are not echoed to the screen.

Repeat()

Prompts the user to rerun or quit and fetches the user's response without echo to the screen. (The user's response is removed from the input buffer.)

WaitToClearScreen()

Displays and horizontally centers (with optional color scheme) "Press any key or mouse button to continue" (or quit) and waits for the user to press a key or mouse button before clearing the screen. Standard and extended keys are recognized. (Default placement is at the bottom (screen row 25) of an 80x25 console screen.

By default, the function will display the message "Press any key or mouse button to continue" (the second argument is passed the symbolic constant CONTINUE_MSG). By passing the symbolic constant QUIT_MSG, the function will display the message "Press any key or mouse button to quit".

ShowStatusBar()

Displays a customized status bar on a specified row. The default screen row is 25, the starting column for displaying a status message is 1, and the status bar color is RED on WHITE.

Console Graphics functions:

setConsoleBoxParams()

Sets the starting X,Y screen drawing coordinates, width, and height of the internal console BOX structure variable that stores information about the current settings of the last drawn console box. (NOTE: Can be used to make DrawBorder() independent of DrawBox().)

getConsoleBoxX()

Returns the starting X screen drawing coordinate of the internal console BOX structure variable that stores information about the current settings of the last drawn console box.

getConsoleBoxY()

Returns the starting Y screen drawing coordinate of the internal console BOX structure variable that stores information about the current settings of the last drawn console box.

getConsoleBoxW()

Returns the width of the internal console BOX structure variable that stores information about the current settings of the last drawn console box.

getConsoleBoxH()

Returns the height of the internal console BOX structure variable that stores information about the current settings of the last drawn console box.

getScreenBufferWidth()

Returns the width of the screen-buffer as a screen column coordinate.

getScreenBufferHeight()

Returns the height of the screen-buffer as a screen row coordinate.

WriteText()

Displays a single-line string of text in a specified color at a specified screen location. Centering, blinking, and animation options are available.

DrawBox()

Sizes, draws, and fills a console box. Centering, color, and fill character options are available. The smallest box size is 1x1.

DrawBorder()

Draws a double or single line border in a specified color. (NOTE: If used after a call to DrawBox(), the drawn border reduces the inner dimensions (width and height) of the console box by two character cells. If you want to maintain the original dimensions of the console box, increase the width and height by two when calling DrawBox().)

FillConsoleAttribute()

Changes the attribute (Background/Foreground color) of a specified number of character cells.

SaveScreen()

Saves the character and attribute (color) values of a group of contiguous screen locations. By default, the entire contents of the screen-buffer is saved to an internal temporary screen buffer. Alternatively, the contents of a group of contiguous screen locations can be saved. Current cursor position information is saved. (NOTE: To restore the previous screen contents or the contents of a group of screen locations, call RestoreScreen(). Only one screen of data or group of screen locations can be saved at a time. The screen contents saved with this function are retained between multiple calls to RestoreScreen(). To save new screen contents, call DeleteScreen() before calling SaveScreen.)

RestoreScreen()

Restores the character and attribute (color) values of a group of contiguous screen locations saved by a previous call to SaveScreen(). (NOTE: Only one screen can be saved and restored at a time. Current cursor position information saved during a call to SaveScreen() is restored.)

DeleteScreen()

Deletes the internal memory associated with a previous call to SaveScreen(). Call this function if you want to save new screen contents with SaveScreen().

setOffScreenBuffer()

Enables one-of-three possible offscreen console buffers or the default console screen buffer for drawing. This function must be called prior to using any of the class library functions with any one of the pre-defined offscreen buffers. All offscreen buffers are invisible by default although drawing operations to these buffers remain valid.

setVisibleScreenBuffer()

Sets the visibility of the default console screen buffer or one of the three offscreen console output buffers. Call this method to make an offscreen buffer visible or to restore the visibility of the default console screen.

ShowMouseCoord()

Displays the current mouse coordinates at a specified screen row and column. The mouse coordinates are displayed in a specified color on a specified colored graphics bar (with height equal to one row) that spans a specified screen width. (The default screen row and column is (1,1), the colored graphics bar is displayed in WHITE spanning the entire screen width, and the mouse coordinates are displayed in RED.)

Extended Keyboard and Mouse functions:

ReadConsoleInputBuffer()

Reads standard keystrokes from the console input buffer and displays them to the console screen in a specified color. (The input buffer is automatically flushed each time this function executes.) (NOTE: The function works correctly with the backspace and tab keys.)

checkKeyMouseStatus()

Indicates whether a keyboard event or mouse event has occurred. After a call to this function, obtain the event type from getEventStatus() before using getVirtualKey(), getAsciiKey(), getControlKeyState(), getMousePositionX(), getMousePositionY(), and/or getMouseButtonState() to determine what key or combination of keys were pressed, current mouse position, and the state of the mouse buttons. (Keystrokes are not echoed to the screen and are removed from the input buffer.) (NOTE: To prevent erratic behavior, always use this function in a loop and always check its return value before using the associated key and mouse get functions.)

getVirtualKey()

Returns a system-defined virtual key code for any key pressed during a previous call to checkKeyMouseStatus().

getAsciiKey()

Returns the ASCII character code for a standard key pressed during a previous call to checkKeyMouseStatus(). (If an extended key is pressed, a 0 is returned.)

getControlKeyState()

Returns the state of a Control key pressed during a previous call to checkKeyMouseStatus(). This function can be used to differentiate between the left and right ALT keys or CTRL keys and the status of the CAPS LOCK or NUM LOCK keys. This function can also be used in conjunction with getVirtualKey(), getAsciiKey(), getMousePositionX(), getMousePositionY(), or getMouseButtonState() to determine key or key/mouse combinations. The following symbolic constants are available:

CAPSLOCK_ON	The caps lock light is on.
ENHANCED_KEY	The key is enhanced.
LEFT_ALT_PRESSED	The left alt key is pressed.
LEFT_CTRL_PRESSED	The left ctrl key is pressed.
NUMLOCK_ON	The num lock light is on.
RIGHT_ALT_PRESSED	The right alt key is pressed.
RIGHT_CTRL_PRESSED	The right ctrl key is pressed.
SCROLLLOCK_ON	The scroll lock light is on.
SHIFT_PRESSED	The shift key is pressed.

getMousePosX()

Returns the X screen coordinate of the mouse cursor during a previous call to checkKeyMouseStatus().

getMousePosY()

Returns the Y screen coordinate of the mouse cursor during a previous call to checkKeyMouseStatus().

getMouseButtonState()

Returns the state of the mouse buttons during a previous call to checkKeyMouseStatus().

The following constants are defined for the first five mouse buttons:

FROM_LEFT_1ST_BUTTON_PRESSED
RIGHTMOST_BUTTON_PRESSED
FROM_LEFT_2ND_BUTTON_PRESSED
FROM_LEFT_3RD_BUTTON_PRESSED
FROM_LEFT_4TH_BUTTON_PRESSED

getEventStatus()

Returns the console-input buffer 'event status' return value from a previous call to checkKeyMouseStatus().

getMouseEventType()

Returns the mouse event type from a previous call to checkKeyMouseStatus(). If this value is zero, it indicates a mouse button being pressed or released. Otherwise, the value is one of the following:

Value	Meaning
DOUBLE_CLICK	The second click (button press) of a double-click occurred. The first click is returned as a regular button-press event.
MOUSE_MOVED	A change in mouse position occurred.
MOUSE_WHEELED	The mouse wheel was rolled.

WaitForAnyKeyPress()

Polls the keyboard waiting for any standard or extended key to be pressed. (The function only returns when a key is pressed.)

WaitForMouseClick()

Polls the mouse waiting for a mouse button to be pressed. (The function only returns when a mouse button is pressed.)

PollKbdMouse()
  Polls the keyboard and mouse waiting for a standard key, extended key, or mouse button to be pressed. (The function only returns when a key or mouse button is pressed.)

CCL2 has the following limitations:

- Multiple screen buffers are supported but only one can be visible at a time.
- Screen-buffer sizes larger but not smaller than 80x25 are supported.
- Automatic scrollbars can be enabled for screen-buffer sizes larger than 80x25 but must be set through Windows system settings (Win98 only).
- Only color text-mode is supported. Graphics modes are not supported.
- Only 16 Foreground colors and 16 Background colors are available.

III. Using CCL2 In The Classroom

A typical approach to using CCL2 in the classroom is to have students use some of the member functions in the library in their own programs without knowing how those functions are written. Once students become comfortable with using the library, instructors can select the source code of particular library functions to examine in class or as homework exercises. In this way, instructors can use the library implementation to supplement the instruction of basic C/C++ concepts and techniques.

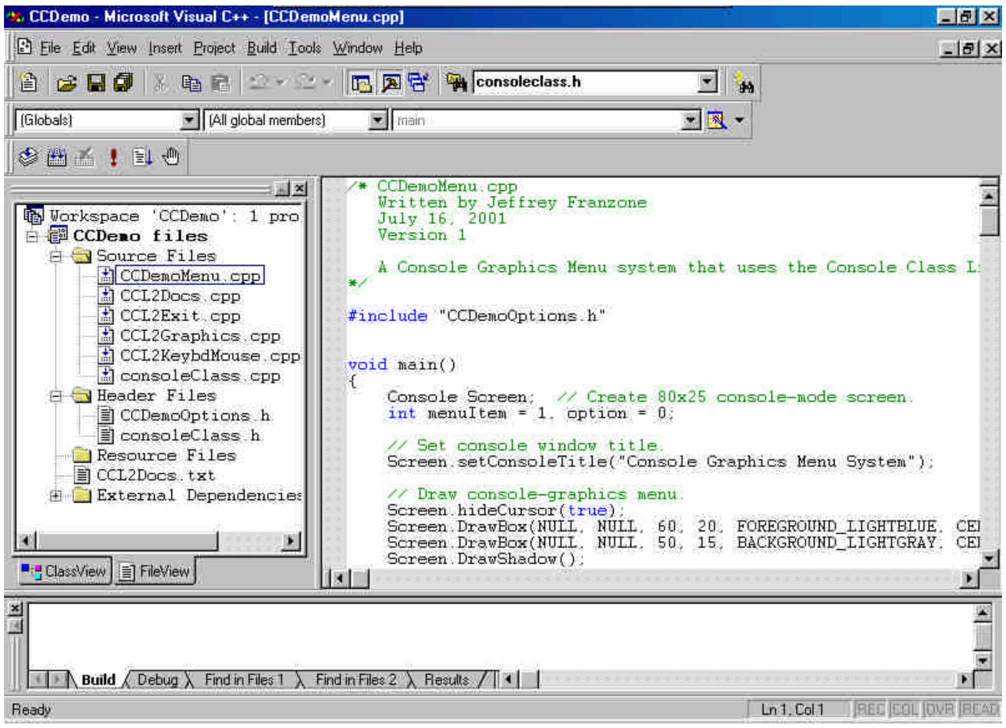
The CCL2 library is free and available in a download package (refer to the *Bibliography* for an email contact). It consists of the CCL2 library files, useful documentation that provides insights into the design and operation of CCL2, and a Demo Menu program with associated demo files.

The demo files provided in the download package demonstrate many of the features of CCL2 and provide additional source-code showing students how to incorporate these features into their own code. The demo files are organized around a console-mode menu system (sometimes called a text menu) in which the user can select the desired demo. To run the CCL2 demos, do the following:

1. Create a new Win32 Console Application in Visual C++ 6.0/7.0.
2. Using the *"Add to Project"* command under the *Project* menu, add the following files to the project. You should have a screen similar to the one shown in Figure 1.

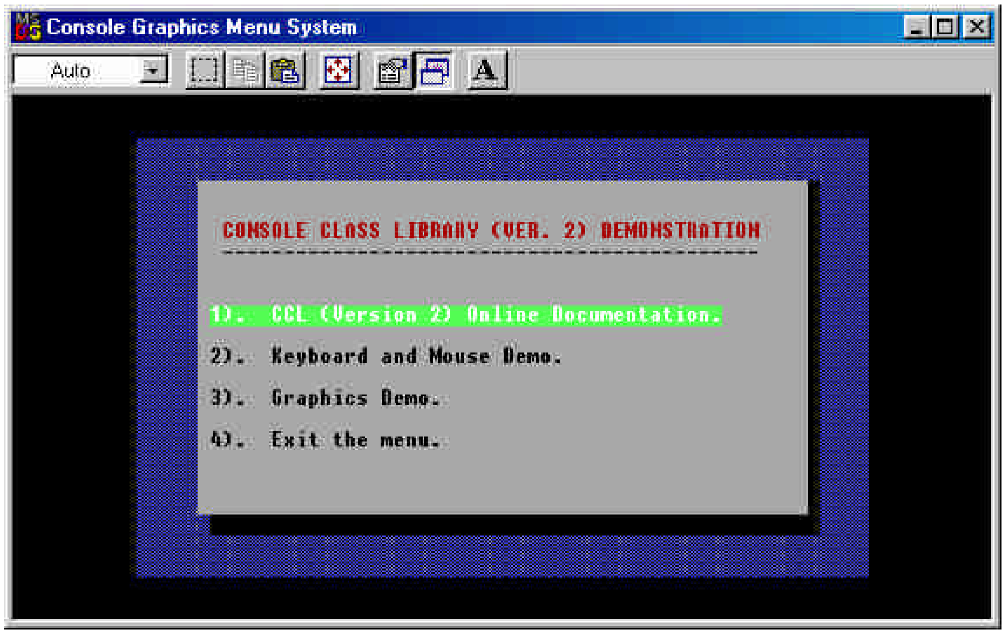
a.	CCDemoMenu.cpp	// Console-menu interface.
b.	CCL2Docs.cpp	// Online documentation demo.
c.	CCL2Docs.txt	// Text file read by CCL2Docs.cpp.
d.	CCL2KeybdMouse.cpp	// Console keyboard and mouse demo.
e.	CCL2Graphics.cpp	// Console graphics demo.
f.	CCL2Exit.cpp	// Sign-off screen demo.
g.	CCDemoOptions.h	// Include file for all demo programs.
h.	consoleClass.cpp	// CCL2 source file.
i.	consoleClass.h	// CCL2 include file.

3. Build the project and execute the demo file (Select the *"Execute…"* command from the Build menu). You should initially see a console-mode menu system (called the *"Console Graphics Menu System"*) as shown in Figure 2. Use the keyboard or mouse to select the desired demo program. Note that when a demo program is finished, it returns to the menu system. To exit the menu system, select the exit option.



```
/* CCDemoMenu.cpp
   Written by Jeffrey Franzone
   July 16, 2001
   Version 1

   A Console Graphics Menu system that uses the Console Class L
*/

#include "CCDemoOptions.h"

void main()
{
    Console Screen;  // Create 80x25 console-mode screen.
    int menuItem = 1, option = 0;

    // Set console window title.
    Screen.setConsoleTitle("Console Graphics Menu System");

    // Draw console-graphics menu.
    Screen.hideCursor(true);
    Screen.DrawBox(NULL, NULL, 60, 20, FOREGROUND_LIGHTBLUE, CE
    Screen.DrawBox(NULL, NULL, 50, 15, BACKGROUND_LIGHTGRAY, CE
    Screen.DrawShadow();
```

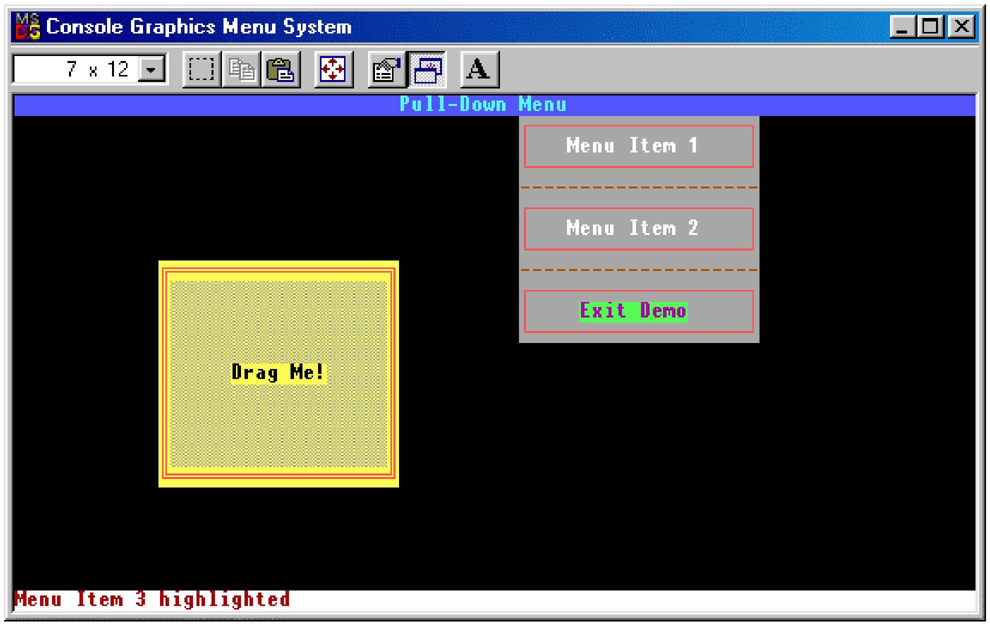
FIGURE 1. Adding the CCL2 demo program files into a new project.



```
CONSOLE CLASS LIBRARY (VER. 2) DEMONSTRATION
--------------------------------------------

1). CCL (Version 2) Online Documentation.

2). Keyboard and Mouse Demo.

3). Graphics Demo.

4). Exit the menu.
```

FIGURE 2. The CCL2 "Console Graphics Menu System".

The CCL2 Demo Menu program consists of three demonstration programs: *Online Library Documentation, Keyboard and Mouse Demo, and Graphics Demo*. The following screenshots show some of CCL2's capabilities.



Menu Item 3 highlighted

FIGURE 3. Graphics Demo Screenshot.

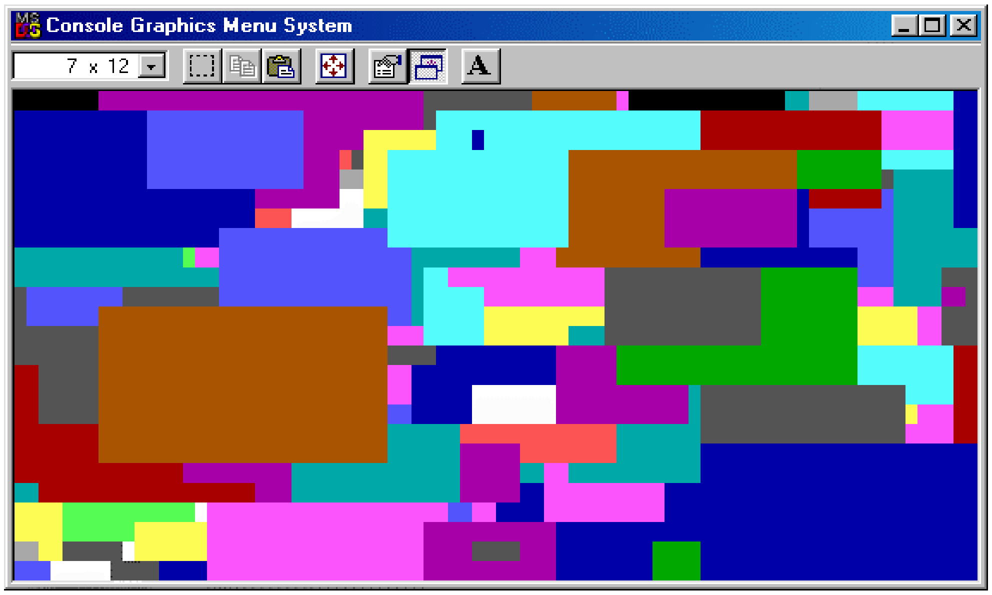
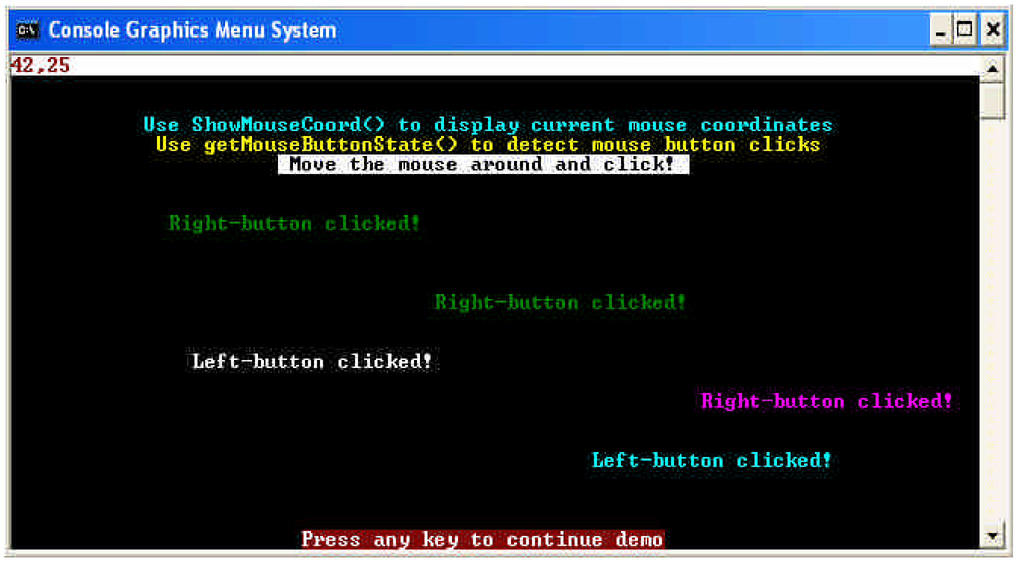
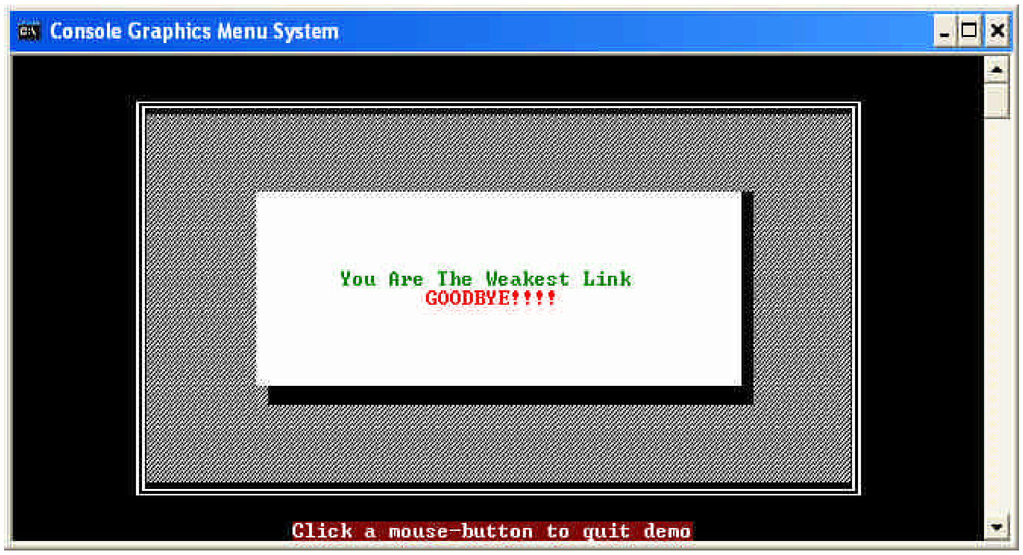


FIGURE 4. Graphics Demo Screenshot.



```
42,25

              Use ShowMouseCoord() to display current mouse coordinates
               Use getMouseButtonState() to detect mouse button clicks
                            Move the mouse around and click!


                Right-button clicked!


                                      Right-button clicked!


                  Left-button clicked!

                                                          Right-button clicked!


                                                  Left-button clicked!


                              Press any key to continue demo
```

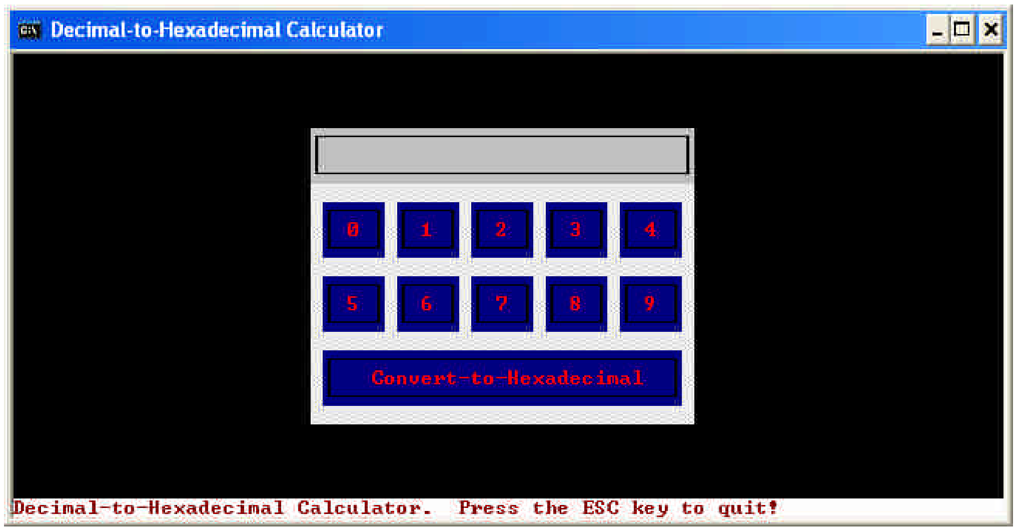
FIGURE 5. Keyboard and Mouse Demo Screenshot.



```
You Are The Weakest Link
       GOODBYE!!!!

Click a mouse-button to quit demo
```

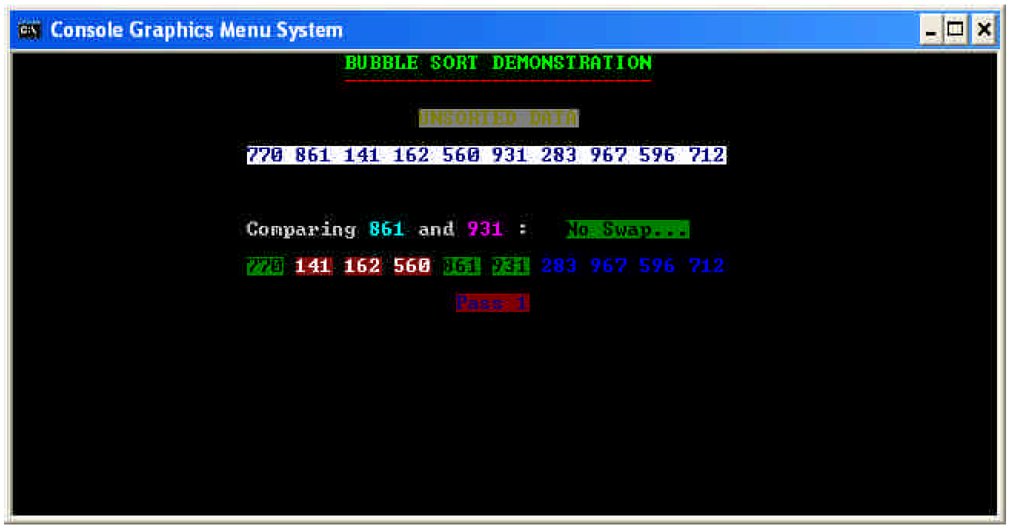
FIGURE 5. Keyboard and Mouse Demo Screenshot.

The following screenshots are taken from actual student programs using CCL2.



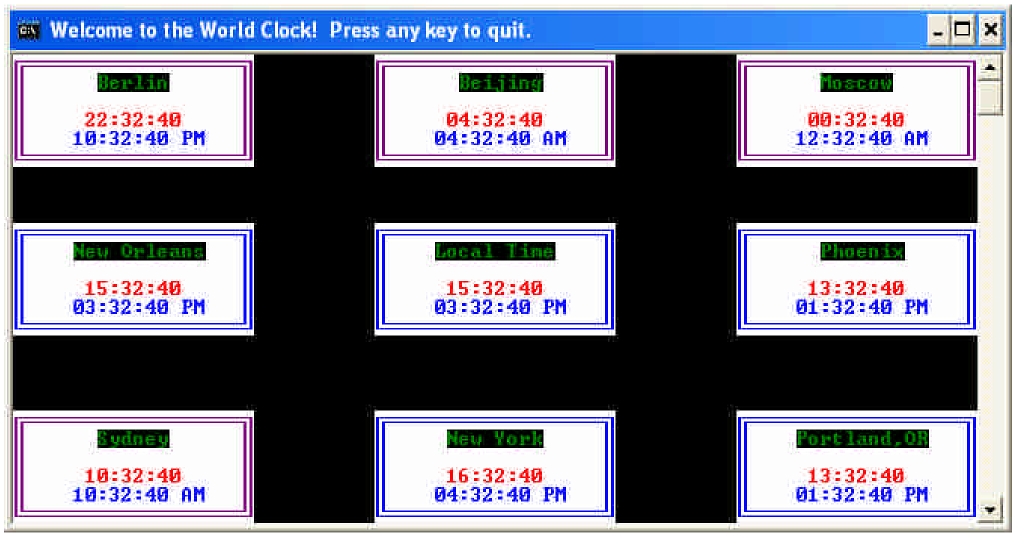
```
Decimal-to-Hexadecimal Calculator.  Press the ESC key to quit!
```

FIGURE 6. A Decimal-to-Hexadecimal Calculator.



```
BUBBLE SORT DEMONSTRATION

UNSORTED DATA

770 861 141 162 560 931 283 967 596 712


Comparing 861 and 931 :   No Swap...

770 141 162 560 861 931 283 967 596 712

Pass 1
```

FIGURE 7. A Bubble Sort Demonstration Program.



```
Welcome to the World Clock!  Press any key to quit.

   Berlin            Beijing           Moscow
  22:32:40          04:32:40          00:32:40
10:32:40 PM       04:32:40 AM       12:32:40 AM

 New Orleans        Local Time        Phoenix
  15:32:40          15:32:40          13:32:40
03:32:40 PM       03:32:40 PM       01:32:40 PM

   Sydney           New York        Portland,OR
  10:32:40          16:32:40          13:32:40
10:32:40 AM       04:32:40 PM       01:32:40 PM
```

FIGURE 8. World Clock Program.

IV. Conclusion

CCL2 provides a rich collection of console mode functions that increase the functionality and embellishment of Win32 console mode programs. The library acts as a bridge between traditional non-GUI programming techniques typically taught in lower-division programming courses and advanced GUI programming techniques typically taught in upper-division programming courses. When used as a teaching tool in lower-division C++ courses, students learn to appreciate the mechanics of GUI programming while incorporating simple GUI elements into their programs. Because all of the source code is freely available, instructors can use the library as an aid in helping students understand basic C++ concepts by selecting which member functions to examine in detail as class or homework exercises. As a programming instructor, you are sure to spark some interest in your students as they use CCL2 in their programs.

Bibliography

1. Microsoft Visual C++ 6.0 contained in Microsoft Visual Studio 6.0, Professional Edition, 1998, Microsoft Corporation.
2. MSDN Library Visual Studio 6.0, 1998, Microsoft Corporation.
3. To request a free CCL2 download package, contact the author through email at jfranzon@mem.quik.com

JEFFREY FRANZONE
Jeffrey Franzone currently teaches in the Engineering Technology Department at the University of Memphis as an Assistant Professor. He teaches C, C++, Java, and microprocessor courses. He has 7 years industrial experience working as an engineering technologist both in hardware and software design and testing and 9 years teaching in Engineering Technology. Jeff received a Bachelor's degree in Electronics Engineering Technology at California State University at Long Beach in 1991 and received a Master's degree in Computer Technology at Arizona State University in 1996.