CS 215 MFC Frequently Asked Questions (FAQ)

1. Where can I find CS 215 MFC projects from other semesters?

http://watts.cs.sonoma.edu/CS215Projects/

2. Where can I find MFC coding examples?

http://watts.cs.sonoma.edu/cs215f16/Lab09/index.html

~tiawatts/cs215pickup/StopWatch

Getting Started with Microsoft Visual C++ 6 with an Introduction to MFC
by Harvey M. Deitel, Paul J. Deitel, Tem Nieto and Edward Strassberger

3. How do I add sound to my Visual Studio C++ MFC application?

1. Use Audacity to create a .wav file; the file should be no larger than 1M (1000K).

2. Add a .wav resource to resource.rc

```
SOUND_WAV WAVE "whatever.wav"
```

3. Add your .wav file to project

Place the .wav file in the same directory as your .h, .cpp, .rc, and .bmp files

4. In properties, under Linker>>Input add winmm.lib to Additional Dependencies

5. Add

```
#include <mmsystem.h>
```

to your file that will call PlaySound (make sure this is after #include <afxwin.h>)

6. Call PlaySound:

```
BOOL soundPlayed = PlaySound(CString("SOUND_WAV"),
       GetModuleHandle(NULL),SND_RESOURCE | SND_ASYNC);
```

Notes - Please make sure that you use the two flags indicated in the above call to PlaySound. If you used SND_FILENAME instead of SND_RESOURCE, the file must be in the same directory system as the executable - this will cause a problem for anyone who wishes to download and run your app. If you do not use SND_ASYNC, you cannot run the music while the game is playing. You may also use SND_LOOP if you wish to loop the music while the game is playing:

```
SND_RESOURCE | SND_ASYNC | SND_LOOP
```

This should work with either DEBUG or RELEASE mode.

4. How do I use multiple fonts in my Visual Studio C++ MFC application?

The default font for MFC is Arial, 12 point. The following steps will create a new font and save the original font so that it can be used again later.

1. Create the line of text that you will be displaying:
```
char message[100];
sprint_s (message, "The AIA Player's score is: %d", aiaScore);
```

2. Create a font:
```
CFont font1;
CString fontName = "Myriad Pro";
int fontSize = 20;
font1.CreatePointFont(fontSize, fontName, &dc);
```

3. Choose a font color:
```
COLORREF def_color = dc.SetTextColor (RGB (255, 60, 60));
```

4. Select the font and save the original font:
```
CFont* def_font = dc.SelectObject(&font1);
```

5. Draw the message at the desired location:
```
CRect textLocation = CRect (50, 70, 80, 90);
dc.DrawText(CString (message), textLocation, DT_CENTER);
```

6. Create and use a second font:
```
CFont font2;
fontName = "Comic Sans MS";
fontSize = 40;
font2.CreatePointFont(fontSize, fontName, &dc);
dc.SelectObject(&font2);
CRect textLocation = CRect (150, 170, 180, 190);
dc.DrawText(CString (message), textLocation, DT_CENTER);
```

7. Restore the original font and color:
```
dc.SelectObject(def_font);
dc.SetTextColor (def_color);
```

5. How do I center text in a rectangle?

1. Create the line of text that you will be displaying:
```
CString message = "This is my message";
```

2. Create the rectangle for the text:
```
CRect textLocation = CRect (50, 70, 80, 90);
```

The default location for the text is the upper left corner of the rectangle.

3. To print the text in the horizontal center of the rectangle, use the DT_CENTER flag:
```
dc.DrawText(message, textLocation, DT_CENTER);
```

4. To print the text in the vertical center of the rectangle, use the DT_SINGLELINE and DT_VCENTER flags:
```
dc.DrawText(message, textLocation, DT_SINGLELINE | DT_CENTER);
```

5. To print the text in the horizontal and vertical center of the rectangle, use all three flags:
```
dc.DrawText(message, textLocation,
                        DT_CENTER | DT_SINGLELINE | DT_CENTER);
```

6. How do I print variable values in my text?

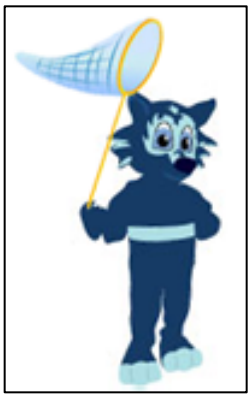
There are 2 ways to do this: using C-style stdio or using a C++ stringstream.

1. Using C-style stdio:
   a. Include cstdio and use the standard namespace:
```
#include <cstdio>
using namespace std;
```
   b. Declare your variable in your class, for example:
```
int score;
```
   c. Create a C-style string buffer long enough to hold your message:
```
char buffer [100];
```
   d. Use sprintf_s to write a formatted message to the buffer, for example:
```
sprintf_s (buffer, "Your score is %d", score);
```
   e. Copy the buffer to a MFC CString:
```
CString message = buffer;
```

2. Using a C++ stringstream:
   a. Include sstream and use the standard namespace:
```
#include <sstream>
using namespace std;
```
   b. Declare your variable in your class, for example:
```
int score;
```
   c. Create an output stringstream buffer:
```
ostringstream buffer;
```
   d. Use the output operator (<<) to write a message to the buffer, for example:
```
buffer << "Your score is " << score;
```
   e. Copy the buffer to a MFC CString:
```
CString message = buffer.str();
```

7. How do I use PlgBlt to mask and rotate an image?

1. Create a Bitmap Image (.bmp) and an identically sized black and white Bitmap mask image:



The black part of the mask will prevent the selected pixels from being drawn by the PlgBlt command. (These images are both 100 x 160)

2. Add both images to your project and your .rc file, for example:

```
LOBOR_BMP        BITMAP     LoboR.bmp
LOBORMASK_BMP    BITMAP     LoboRMask.bmp
```

3. Load the bitmaps into your image array, for example:

```
res = images[MEimg].LoadBitmap(CString("LOBOR_BMP"));
res = images[MEimg+1].LoadBitmap(CString("LOBORMASK_BMP"));
```

4. Determine the location of the window where you wish to draw the image, for example:

```
CRect location = grid[currentRow][currentCol].where;
```

5. Create an array of three points to define the parallelogram in which the image will be drawn:

```
CPoint points [3];
points[0] = location.TopLeft();
points[1] = CPoint (location.right, location.top);
points[2] = CPoint (location.left, location.bottom);
```

6. Call the PlgBlt command, for example:

```
deviceContextP->PlgBlt(points, &memDC,
                       0, 0, 100, 160, images[MEimg+1], 0, 0);
```

Additional notes:

MEimg is the index in the array Images where the first Lobo image is stored. MEimg+1 is the next index value.

0, 0, 100, 160 are the top left and bottom right coordinates of the portion of the Lobo image being drawn. In this case, it is the whole image. The last 0, 0 is the corresponding top left coordinate of the mask image.