**IN628 2019 Practical 01.2 – Components Review**

1. **Exercise 1**

Create a ***Form*** with a ***Button***. Each time the ***Button*** is clicked, increase the ***Button’s*** width by 10 pixels. When the right edge of the ***Button*** reaches the edge of the form, change the text of the button to “Too Big”.

1. **Exercise 2**

Create a ***Form*** with a ***Button*** and ***TextBox***. When the ***Button*** is clicked, its text changes to equal the contents of the text box.

1. **Exercise 3**

Create a ***Form*** with a ***Button*** and ***ListBox***. Each time the ***Button*** is clicked, add a new line to the ***ListBox***. The line should read “This is line *n*”, where n is the line number.

1. **Exercise 4**

Create a ***Form*** with a ***Button***. When the ***Button*** is clicked, its text toggles between “On” and “Off”.

1. **Exercise 5**

Create a Form with a ***Button***, ***TextBox*** and ***PictureBox***. Using the Property Inspector, set the BackColor of the ***PictureBox*** to something other than grey, so that it is clearly visible. When the ***Button*** is clicked, the picture box moves n steps to the right, where n is the number entered in the ***TextBox***. Each step should be 10 pixels. After each step, the program should pause for 100 milliseconds (otherwise the ***PictureBox*** moves so fast you can’t see it). To pause for n milliseconds, use the command System::Threading::Thread::Sleep(n).

1. **Exercise 6**

In this problem, you will dynamically change the image contents of a ***PictureBox*** control. To do this, you modify the PictureBox->Image property. You set it to an Image object, which is created by calling Image::FromFile(“filename”). For example, you can display the picture file chicken.jpg as follows:

pictureBox1->Image = Image::FromFile(“chicken.jpg”);

You have been provided a set of 9 images called Dragon1.bmp, Dragon2.bmp, etc. Create a Form with a ***Button*** and a ***PictureBox***. When the ***Button*** is clicked use technique described above to load the ***PictureBox*** with each of the dragon pictures in order from 1 to 9 to produce a simple animation. Some technical points:

1. This problem can be solved by brute force, or more elegantly by using a for loop and the string concatenation operator “+”. Strive for elegance whenever possible.
2. Use relative paths for the images; never hard-code a drive into your file paths. Relative paths in C++ CLI start from the project folder (i.e. where the Form's .h and .cpp files are found), not the Solution folder or the Debug folder.
3. Use Sleep(n) as above to control the timing, and use Application::DoEvents() to refresh the screen after sleeping.
4. Set the ***PictureBox's*** SizeMode property to allow it to dynamically size to its contents. This lets you change the image set without having to recompile.
5. **Exercise 7**

C++ CLI applications do not automatically provide a canvas. Instead, you must create a Graphicsobject to paint on. To create such an object, use the following code:

Graphics^ mainCanvas = CreateGraphics();

You must be using namespace System::Drawing to have access to the Graphics class.

Note that only some C++ CLI classes expose the CreateGraphics() method. The Form is one of them. So, you can create a Graphics object as shown above in any Form method (e.g. in the Form\_Load or in any button click handler).

After you have created a Graphics object you have many methods for drawing on it. You can draw shapes, images, text, lines, etc. Today we will use only one of the many Graphics methods: FillEllipse.

The syntax for FillEllipse is: FillEllipse(Brush, x, y, width, height)

The parameters x and y are the coordinates of the upper left corner of the ellipse, and the parameters width and height are its horizontal and vertical dimensions. The Brush is another member of the System::Drawing namespace. Brushes determine the colour inside a drawn shape. (There is another object Pen that determines the colour of the outline of a shape.) Before calling FillEllipse, you must create a Brush. The full syntax to create a SolidBrush (there are other varieties we will see during the term) is:

Brush^ greenBrush = gcnew SolidBrush(Color::Green);

Given the above discussion, what do you think happens when the button shown below is clicked? Test it and see.

private: System::Void button7\_Click(System::Object^ sender, System::EventArgs^ e)

{

Graphics^ mainCanvas = CreateGraphics();

Brush^ greenBrush = gcnew SolidBrush(Color::Green);

mainCanvas->FillEllipse(greenBrush, 100,100,10, 10);

}

Using the techniques just described, make the following: Create a ***Form*** with a ***Button*** and ***Timer***. At each ***Timer*** tick, draw an ellipse at a random location on the screen. The ellipse should be of random width between 0 and 100 pixels, and random height between 0 and 100 pixels. Arrange to have approximately equal proportions of red, green, blue and yellow ellipses. Include a ***Button*** that turns the ***Timer*** on and off.