Courses 2DCx: 2D-Computer Graphics with C++/MFC
Chapter C1: Comments to the Intro Project

In `CChildView.cpp` inside `void CChildView::OnPaint()`

`dc.TextOut(10,10, "Hello world, here is Intro1 !");` //This statement writes a line of black text into the left upper corner of your program window. The letters `dc.` in front of `TextOut` identify your own window. The parameters `10, 10` indicate the x and y coordinates in pixels where the text should start. Try out other cooerdinates such as `0, 0` and `10,100` in order to become familiar with the coordiante system.Its starting point is the left upper corner of the client area of your program window. The client area is the inner white rectangle that can be used for drawings.

`CRect r; //` Declares a data structure af type `CRect` describing an axis-parallel rectangle named r by 4 integers: r.left, r.top, r.right, r.bottom.

`CString sometext; //` Declares an empty data structure af type CString which is intended to contain any text of arbitrary length.

`GetClientRect( r ); //` A Windows-API-function that asks the operating system to write the values of the client area of the current window into `CRect r`.

`sometext.Format( "width=%d, height=%d", r.right, r.bottom ); //` Format is a member-function of `CString` that combines arbitrary text and text converted integer values into one common string.

`dc.TextOut(10,30, sometext );` //Writes a line of black text underneath the existing line `"Hello world, here is Intro1 !"`

`dc.SetTextColor(RGB(255,0,0));` //`RGB(...)` is a macro that unites 3 bytes into one `COLORREF` value. The first byte indicates the amount of red color, the second the amount of green color and the third the amount of blue color (see lecture Color Models, The RGB Color Model). The statement influences the `TextOut` statements that follow and has no has no immediate consequences. They will draw their texts in red. Change the parameters of RGB into `(0,255,0)`, `(0,0,255)` and to arbitrary combinations such as `(55,255,127).`

`dc.TextOut(10,50, "Change the size of your window !" );` //Draws a red third line below the lines that already exist.

`CPoint P;` //Declares a data structure af type `CPoint` describing a 2D-point with the integer coordinates `p.x` and `p.y`.

`p.x = r.right / 2;` //Fills `p.x` with the half width of the client area.

`p.x = r.bottom / 2;` //Fills `p.y` with the half height of the client area. `P` is now the middle of the client area.

`dc.SetTextColor(RGB(0,0,255));` //This will cause the following texts to be drawn in blue.

`dc.TextOut( 0 , p.y , "left" );` //Put the string "left" to the middle of the left border.
`dc.TextOut( r.right-50, p.y , "right" );` //Put the string "right" 50 pixels left of to the middle of the right border.
`dc.TextOut( p.x , 0 , "top" );<` //Put the string "top" to the middle of the upper border.
`dc.TextOut( p.x , r.bottom-20, "bottom" );` //Put the string "bottom" 20 pixel above the middle of the lower border.

`dc.MoveTo(0,0); dc.LineTo(r.right,r.bottom); dc.MoveTo(r.right,0); dc.LineTo(0,r.bottom);` //This line contains 4 statements of 4 lines of C-Code. `dc.MoveTo(0,0);` has no immediate effect. It fixes the invisible start point of a series of black dots that simulate a geometric straight line on your screen. From this start point `(0,0)` the next statement `dc.LineTo(r.right,r.bottom);` begins to draw a line from the upper left corner to the lower right corner. Invisible programs inside the operating system (if you have a modern computer probably inside your graphics board) compute all the intermediary black dots between coordinate `(0,0)` and coordinate `(r.right,r.bottom)` and put the dots into the video memory of your graphics board and on the screen. In the same way you draw (more exactly: you simulate drawing by putting single dots close together) a second straight geometric line form the upper right corner back to the lower left. The statements `MoveTo` and `LineTo` are the basic statements of any sort of vector graphics. It is so easy to use them but what they really do behind the curtain of the operating system and the graphic board driver is not simple at all.

`int w5 = r.right / 5;` //w5 is 1/5 of the client's area width.

`int h5 = r.bottom / 5;` //h5 is 1/5 of the client's area height.

`dc.Rectangle( w5, h5, 4*w5, 4*h5 );` //Draws a rectangle

`dc.Ellipse ( w5, h5, 4*w5, 4*h5 );` //Draws an ellipse inside the rectangle.

`#define nn 120` //This is not a normal line of code, it is a preprocessor statement. When you write a `#` in the first column of the of the editor, you tell the compiler that you want to talk to him privatly. With `#define nn 120` you tell him that you are tired to write 120. You want him to accept the string `nn` instead of `120`.This makes sense when you want to stay flexible with some constant values that occur several times in your code. Whenever you want to replace the constant `120` to `240` you do not need it to change everywhere. You just change it once in this preprocessor definition.
Beware of this frequent mistake: Do not close preprocessor statements with a semicolon !

`int i;` // Definition of an integer named `i`.

`CPen pen;` // Declares a data structure named `pen` of type `CPen`, that allows you to choose, a line style, a line thickness and a line color for drawing.

`CPoint splash[ nn ];`//You reserve memory space for nn coordinates of type `CPoint` and you give the name `splash` (splash this is not a key word, it is an arbitrary name) to the array. each coordinate is of type `CPoint`, a data stucture provided by MFC which contains 2 integers x and y. You can access the x- coordinate of point no. i by writing `splash[i].x`.

`double arcus = 2. * 3.14159 / nn;` //Definition of an angle of 1/120 of a full circle (in radians)

`double radius_x = 1.5 * w5;` //Definition of a maximal horizontal radius

`double radius_y = 1.5 * h5;` //Definition of a maximal vertical radius

`for ( i = 0; i < nn; i++ )` //Do all the enclosed statements that follow 120 times.

`COLORREF multicolor = RGB ( rand()%255, rand()%255, rand()%255 );` //Take 3 random integers from the Windows API random integer generator (in the range between 0 to nearly infinity), divide them by 255 and preserve only the rest (% is the modulo-operation). The resulting values are random values just between 0 and 255. Put them into a RGB-macro to form a random color. Store this color in a variable named "multicolor" of type `COLORREF`.

`pen.CreatePen( PS_SOLID, 20, multicolor );` //Create a pen for solid lines of a thickness of 20 pixels and give it the random color.

`double factor = (double)rand() / (double)RAND_MAX;` //A random no. is divided by `RAND_MAX` , the biggest random no. that the random generator can produce. The result is a random value between 0.0 and 1.0.

`if ( factor < 0.25 ) factor = 0.25;` // We do not want smaller factors than 1/4.

`double cosinus = radius_x * factor * cos( i * arcus );` //this gives a x-value of a point inside a circle.

`double sinus = radius_x * factor * sin( i * arcus );` //this gives a y-value of a point inside a circle.

`dc.MoveTo( p );` // Start point of all lines is the middle point of the client area.

`dc.LineTo( p.x + (int)cosinus, p.y + (int)sinus );` // Draw a digital line to an end point somewhere inside the ellipse.

`pen.DeleteObject();` //Throw away the pen, for the next line we will buy a new one.

`splash[i].x = p.x + int( cosinus * 0.8 ); splash[i].y = p.y + int( sinus * 0.8 );` // Remember the end point by storing it in the point array (a litlle closer to the midpoint) for further use.

`dc.SelectStockObject( WHITE_PEN );` // take a preconstructed white pen from the operating system.

`CBrush brush;` // Declares a data structure named `brush` of type `CBrush`, that allows you to choose a color for painting.

`brush.CreateSolidBrush( RGB( 255,0,0 ) );` // Construct a red brush.

`dc.SelectObject( brush );` //Take it in your hand.

`dc.Polygon( splash, nn );` // Draw a white continuous polygonal line from vertex no. 0 to vertex no. 120 and fill its interior with red color. The Windows API function `Polygon` replace in a convenient way a lot of calls of `MoveTo` and `LineTo`. It accepts infinite amounts of CPoints when they are placed in a linear array of CPoints. The first parameter is the name of the array (which is a pointer on the first CPoint in the array) and the second parameter has to indicate the size of the array. Take care that this parameter is correct, otherwise you will obtain crazy results.

`brush.DeleteObject();` //Throw the brush away and free its space.

`dc.SetTextColor( RGB( 0,0,255 ) );` // in the future we want to write blue text.

`dc.TextOut( p.x-30, p.y-8, "Splash !" );` // Write the word "splash !" in the middle of the client area.

`Sleep( 100 );` //Ask the operating system to stop the current thread for 100 milliseconds. This avoids a hectic animation.

`Invalidate();` //Ask the operating system to raise the Paint event again. When there are no other high priority tasks the operating system will send a `Paint`-Message to our `CChildView`-instance which calls the `void CChildView::OnPaint()`-event-handler-function again. Calling `Invalidate()` from inside the `OnPaint`-event handler results in a sort of recursive loop which infinitely drives the animation. That's a primitive method to produce (flickering) movements. In further chapters better methods with a `Timer`-object and with double buffering will be proposed.