Courses 2DCx, C4: File, Code Comments

In `file1Doc.h` in front of `class CFile1Doc : public CDocument`

`#include < vector >` //Include the declarations of dynamic arrays from the Standard Template Library STL.

In `file1Doc.h` inside `class CFile1Doc : public CDocument`

`std::vector< CPoint > p;` //Declare a dynamic array to store coordinates of type `CPoint`.

In `file1View.cpp` inside `void CFile1View::OnDraw(CDC* pDC)`

This function makes nearly the same as function `OnPaint()` in the former projects but offers a pointer `pDC` to the current device context.

`CFile1Doc* pDoc = GetDocument();` //This line is from Microsoft. Leave it here. It defines a pointer pDoc to the current document class with its dynamic array `p`.

`ASSERT_VALID(pDoc);` //This line is from Microsoft. Leave it here. It is a debug statement that produces an error message if a pointer happens to be NULL.

`pDC->TextOut( 10, 10, "draw->File store->File close->File open" );` //This is a message to an user who does not know what to do after starting the program.

`int n = pDoc->p.size();` //This statement asks the dynamic array `p` to tell its current length.

`if ( n < 2 ) return;` //If there are less than two vertices, we cannot draw anything.

`pDC->Polyline( &(pDoc->p.front()), n);` //If there are more than one vertex in the array we draw a thin black polyline on the client area. &(pDoc->p.front()) is the adress of p[0], the start adress of array p.

In `void CFile1View::OnLButtonDown(UINT nFlags, CPoint point)`

Somebody pressed the left mouse button. This person will probably draw something.

`CFile1Doc* pDoc = GetDocument();` //Look for a pointer to the document class.

`pDoc->p.clear();` //Take the pointer, access the dynamic array p and erase any vertex that may be in. The array is empty now. Its size is now zero. All its memory is free.

`pDoc->p.push_back( point );` //Store the current mouse position as the first vertex. The length of array `p` is again one vertex (= 2 integers x and y).

`Invalidate();` //Clean up the client area from all former drawings.

In `void CFile1View::OnMouseMove(UINT nFlags, CPoint point)`

`if ( !nFlags ) return;` //Ignore any mouse movements when no mouse button is pressed.

`CFile1Doc* pDoc = GetDocument();` //Look for a pointer to the document class.

`pDoc->p.push_back( point );` //Store one more vertex into the dynamic array `p`.

`Invalidate( false );` //Ask the operating system to redraw the client area without claening it.

In `void CFile1View::OnLButtonUp(UINT nFlags, CPoint point)`
The user released the left mouse button. He does not want to draw more vertices.

`Invalidate();` //Ask the operating system to redraw everything. This statement is not necessary here. Nothing happens when you delete it, because the complete polyline is already visible on the screen since the last mouse movement.

In `file1Doc.cpp` inside `void CFile1Doc::Serialize(CArchive& ar)`
This very powerful MFC-function handles any access to the harddisk in a very comfortable way. It frees us from all the complicated processes of file finding, opening, closing and error handling.

`if (ar.IsStoring())` //This line is from Microsoft. Leave it here. It tests if the archive has to be stored to disk or if it has to be read from disk.

`int n = p.size();` //When it has to be stored, we have to know, how many vertices are in the array.

`if ( n < 2 ) return;` //When there are less then two vertices, it makes no sense to store anything.

`ar << n;` //In front of any other information we write the no. of vertices into the file.

`ar.Write( &p.front(), n*sizeof( CPoint ) );` //Then we write the complete array byte by byte.

`else` //We have to read an array from a file.

`int n;` //We need a local integer variable to hold the no. of vertices.

`ar >> n;` //In front of any other inforamtion there must be an integer with the no. vertices.

`p.resize( n );` //Now we reserve dynamic memory for n vertices.

`ar.Read( &p.front(), n*sizeof( CPoint ) );` //Then we read all vertices from the file byte by byte.

`UpdateAllViews( NULL );` //This statement asks the operating system to redraw the contents of all windows on the screen. This makes sure that our function `OnDraw()` is executed and that we see our polyline.

In `file1View.cpp` inside `void CFile1View::OnLButtonUp(UINT nFlags, CPoint point)`
Write the polyline as VML-code into the preexisting file `vml.html`.

`Invalidate();` //This statement was already here.

`CFile1Doc* pDoc = GetDocument();` //Look for a pointer to the document class.

`int n = pDoc->p.size();` //This statement asks the dynamic array `p` to tell its current length.

`if ( n < 2 ) return;` //nothing to do

`CFile vml ( _T("vml.html"), CFile::modeReadWrite );` //Open the exisiting file `vml.html`. If no such file is found in the current path, the program will crash.

`int length = vml.GetLength();` //Ask the file for its length.

`CString s;` //Instance of class `CString`.

`vml.Read ( s.GetBuffer(length), length );` //Read and treat the complete file as one text string.

`int i = s.Find( "/body" );` //Look for a substring `"/body"`.

`if ( i == -1 ) { MessageBeep(-1); return; }` //if `"/body"` not found, forget the rest.

`vml.Seek( i-1, CFile::begin );` //Go back to the head of the file and from there foreward to i-1, which is directly in front of `"/body"`.

`s = "< v:polyline strokecolor =\"red\" strokeweight = \"4px\" points = \"";` //This is VML syntax for polylines.

`vml.Write( s, s.GetLength() );` //Insert the string into `vml.html`.

`for ( i = 0; i < n-1; i++ )` //Take all vertices except the last one.

`s.Format( "%dpx, %dpx, ", pDoc->p[i].x, pDoc->p[i].y );` //Format any vertex into VML syntax.

`vml.Write( s, s.GetLength() );` //Write vertex no. i into `vml.html`.

`s.Format( "%dpx, %dpx\">< /v:polyline >\r\n\r\n< /body >< /html >", pDoc->p[n-1].x, pDoc->p[n-1].y );` //Format the last vertex together with the end polyline tag, two empty lines and the end body- and end html-tag.

`vml.Write( s, s.GetLength() );` //Write this last vertex with adnex to `vml.html`.