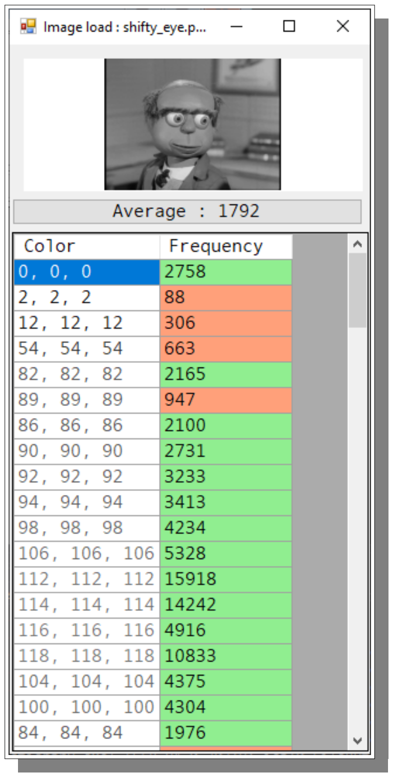
CMPE2300 – ICA10 – Pixel Analyzer

In this ICA you will use Dictionary to create an image analyzer.

Create a new Windows form project for this ica. Construct a UI for your form fashioned after the following capture:



Average : 1792

Color	Frequency
0, 0, 0	2758
2, 2, 2	88
12, 12, 12	306
54, 54, 54	663
82, 82, 82	2165
89, 89, 89	947
86, 86, 86	2100
90, 90, 90	2731
92, 92, 92	3233
94, 94, 94	3413
98, 98, 98	4234
106, 106, 106	5328
112, 112, 112	15918
114, 114, 114	14242
116, 116, 116	4916
118, 118, 118	10833
104, 104, 104	4375
100, 100, 100	4304
84, 84, 84	1976

The UI consists of a Picturebox, a button and a DataGridView anchored to fill the remaining view. Include a initialized BindingSource as a form member.

Add the following fields to your form :

- a field of Dictionary with a key type of Color and value type of int to hold our retrieved colors and their respective frequency values
- an integer representing the current Average frequency value of the dictionary

Form Constructor :

- Assign the DataGridView datasource to the BindingSource.
- Create callbacks for your button (optional Load – see below)
- Create callbacks for the DataGridView ColumnHeaderMouseClick and CellFormatting events.
- Drag/Drop will be used here, so bind the 2 Drag/Drop helper events to your PictureBox.
- Set SizeMode property of the PictureBox to Zoom, this will keep the aspect ratio while maximizing the fill of the image.

Helper function ShowDictionary(), returns a nothing, accepts nothing.

- This function will show the dictionary in the DataGridView. You can just ToList() your dictionary and assign it to the BindingSource. This now creates a List of KeyValuePair elements which will be displayed in the DataGridView. Use Columns[N] property of the DataGridView to set the HeaderText of the columns, and interestingly the key Color is automatically transformed into the r,g,b values*. Before colorizing the view and adding sort functionality, continue coding the Load File functionality.
- Using the Average<> extension method and a lambda, extract the current average frequency from your dictionary. Use this value to update the text of the button on the form as shown and assign the Average form member.
- Now iterating through the dictionary using foreach(), determine the sum of all the frequency values. While an extension and lambda could work here, we want an explicit example of dictionary iteration.

Helper function LoadFile(), accepting a FileInfo object, the image to load. Using Bitmap.FromFile() attempt to create a new Bitmap object – use the returned image object as the CTOR argument for the new Bitmap – why ? Assign the Image of the PictureBox to the new Bitmap. Any error on load should set the background of the PictureBox to Red. Update the form text to include the successful filename “Loaded Image : file_name”. (just the name, not path).

- Clear your dictionary.
- Then iterate through the bitmap retrieving each pixel :
  - if the color doesn't exist in our dictionary, add it with a frequency of zero
  - increment the dictionary at the color – it will be in the dictionary – right ?
  - ** Or, if you noticed, you could use an if/else scenario here, Merits of each ?
- Once you have loaded the dictionary, call your helper to display it in the DataGridView.

Now include colorizing.

In your DataGridView CellFormatting event callback, we will examine each cell as it is rendered. The “e” argument has a ColumnIndex property to indicate which column(0 - Indexed !) this data belongs to.

- The first column should have the background set to the “value” (our color – a cast is required). ** This could become hard to read – what could you do with the forecolor ?
- The 2nd column , Compare if the Value property is less than the Average then set the e.CellStyle.BackColor property to LightSalmon, otherwise set the color to LightGreen. Now as each cell is passed through this event callback it will alter it's color. The Value property is of type object and can be cast to the known type (int). Verify this works. Now frequencies are showing if they have values above or below the average.

Functionality of the [Average] button is to use lambdas with a Where() combined with ToDictionary() calls to remove all dictionary entries that are **less** than the current frequency average. Use the current calculated average value as a constraint for your Where() call. You shall save the resulting dictionary back to your member, and invoke your ShowDictionary() helper to verify the modification. The effect over repeating executions is deleting all below average frequencies, until only one is left.

Now include Sorting of our Data.

Our ColumnHeaderMouseClick callback will now force our data to be either sorted by key or value. To do this, alas, Dictionary doesn't support sort…

We will conquer this using 2 different approaches :

If the “e” callback argument indicates a ColumnIndex of 0, we want Color Ordering :
For the first Color sort :

- use the OrderBy() extension method to return a IEnumerable collection that is ordered by the Color.ToArgb() value (peek at the Func() that OrderBy wants .. return the element to sort by)
- then to turn it back into a Dictionary we can re-assign to our container use ToDictionary()

and specify the appropriate elements. Reassign back to your dictionary member the ordered dictionary.

For the Frequency Column Sort

If the "e" callback argument indicates a ColumnIndex of 1, we want a 2-tier sort with Color within Frequency : (Why would a Frequency within Color be futile ?)

For our second frequency sort, we want a Color within frequency order

- Make a temporary local List of KVP type, then ToList() your dictionary.
- Sort() your list via a nice lambda against the KVP items
- then finally ToDictionary() it back using a nicer lambda specifying the key and value selectors
- *Remember to use the version of ToDictionary() that allows you to specify both key and value selectors. It will populate your original dictionary again - but in the order from your list. This is safe, since we are not changing the key to our frequency which would conflict (potentially) having multiple bytes that might have the same frequency value.

Note : Recall that the single argument version of ToDictionary() will take in entire object type making it the key element - so if used, our dictionary would have a Key of KVP and no value component - that is NOT compatible with our ShowDictionary() method... And more importantly - Dictionary requires the key be unique.. so attempting to populate with key = frequency would likely result in duplicate frequencies resulting in an exception.

Thats it, now test your program on files you can verify (ie make a text file or 2 with known values). The supplied file values are known only to your instructor for checkoff.

Base completion without Sorting or Color coding in DataGridView. 50%
Completion with Sorting and Color coding 20%

LinkedList Enhancement (30%).

This enhancement will include a CDrawer to display the contents of a LinkedList generated from the dictionary. Add a CDrawer form member and initialize in the form CTOR (defaults, continuous update off).

Whenever a new image is loaded, after the ShowDictionary(), invoke a new function ShowLL().

Helper function ShowLL(), accepts nothing, returns nothing. There are two steps in this function, first manually construct a LinkedList of Color from the dictionary, then iterate your linked list and display the colors as blocks in your CDrawer.

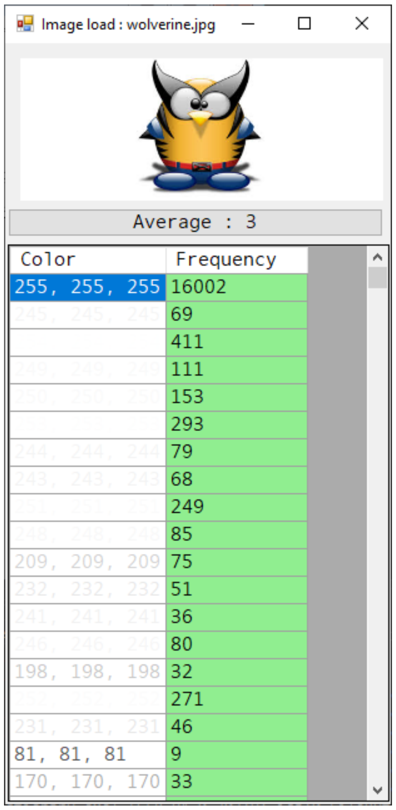
Starting with an empty LinkedList of Color. Using foreach() and KeyValuePair<>, iterate your dictionary, taking a key Color.

- Using a LinkedListNode and a loop, iterate your LinkedList to find the appropriate place to add the Color to maintain a descending ToArgb() value.
- Once the proper place is determined, use AddXXX() method to insert the new Color.

Once the LinkedList is complete, we want to display the contents in the CDrawer in the largest scale value still allowing us to view all nodes.

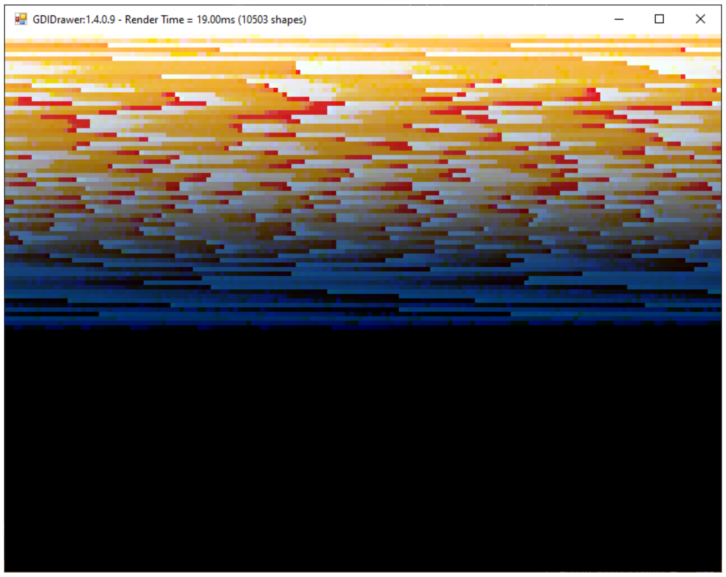
- Clear your CDrawer
- Using a loop, and starting the Scale at 1, iteratively increase the scale until the LinkedList count no longer fits in the scaled area of the CDrawer. You are too big, back off to get the last valid Scale value.
- Using LinkedListNode<> and a for() or while() loop and a separate node counter, add a Rectangle of size 1, typewriter style – wrapping on the width, as we have done before.
- Render

You should now see all the colors retrieved from your dictionary. It is not foolproof, and possible to load an image with too many colors. You do not need to mitigate this.. but what might you do ?



Average : 3

Color	Frequency
255, 255, 255	16002
245, 245, 245	69
[illegible]	411
249, 249, 249	111
250, 250, 250	153
253, 253, 253	293
244, 244, 244	79
243, 243, 243	68
251, 251, 251	249
248, 248, 248	85
209, 209, 209	75
232, 232, 232	51
241, 241, 241	36
246, 246, 246	80
198, 198, 198	32
252, 252, 252	271
231, 231, 231	46
81, 81, 81	9
170, 170, 170	33



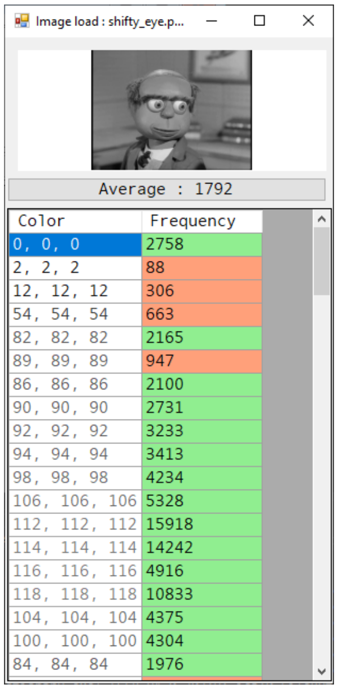
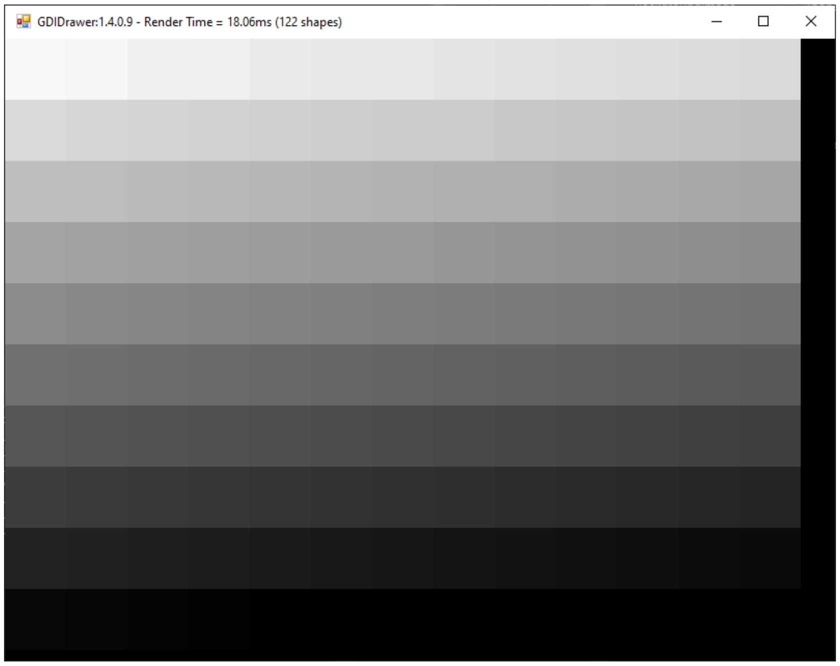


Image load : shifty_eye.p...

Average : 1792

Color	Frequency
0, 0, 0	2758
2, 2, 2	88
12, 12, 12	306
54, 54, 54	663
82, 82, 82	2165
89, 89, 89	947
86, 86, 86	2100
90, 90, 90	2731
92, 92, 92	3233
94, 94, 94	3413
98, 98, 98	4234
106, 106, 106	5328
112, 112, 112	15918
114, 114, 114	14242
116, 116, 116	4916
118, 118, 118	10833
104, 104, 104	4375
100, 100, 100	4304
84, 84, 84	1976



GDIDrawer:1.4.0.9 - Render Time = 18.06ms (122 shapes)