CMP SCI 4420/5420 **Image Processing and Computer Vision**
Assignment # 1 Due Date: September 09, 2024

Image Browser

Purpose

This project leads to the development of an image browser. Given a directory, display each picture in the directory as well as in its subdirectories. The learning objectives in the project include: getting familiar with OpenCV, read and display an image, interact with a display window, and perform affine transforms on the image.

Task

This project involves creating an image browser. You will develop a simple Graphical User Interface (GUI) for browsing images which are stored in a hierarchically-structured directory tree. Let us call the top-level directory as `dirA`. This directory will contain some images as well as other (sub)directories. Let us call these (sub)directories as `dirA1`, `dirA2`, and so on. Each of these subdirectories in turn will contain some images and may contain other (sub)directories. This (sub)directory structure may extend to an arbitrary number of levels. Your GUI should provide the following functions:

- Enable browsing of images which are stored in a hierarchically structured directory tree.
- In addition to displaying images, your GUI should also display meta-data associated with the images. Available meta-data may include image file name, file path, image file type, image size (number of rows and columns), number of pixels (number of rows x number of columns), and image file size in bytes.
- Navigate to display next or previous image depending on user input.
- Ensure that the image fits within specified pixel dimensions while preserving aspect ratio, if the user has specified those dimensions.

Invoking the solution

Your solution will be invoked using the following command:

```
browser [-h] [-rows=numrows] [-cols=numcols] dir
```

`browser`	Name of your executable
`numrows`	Maximum number of rows in the display window [default: 720]
`numcols`	Maximum number of columns in the display window [default: 1080]
`dir`	Input directory

The parameters in square brackets are optional. You'll notice that the style of specifying options is different from standard Linux specifications. This style is favored by OpenCV using the function `cv::CommandLineParser`.

Suggested implementation steps

1. Parse the command line. You can create your own parser or use the class `CommandLineParser` provided by OpenCV. Each of the optional arguments may have a default value. If the user specifies the option `-h`, print a help message and exit. Otherwise, assign the number of rows and number of columns from user inputs or default values. If you are running on Windows, you can find the maximum screen size and use that as default. The purpose of specifying the maximum number of rows and columns for displays is to provide a box in which the picture must fit.

2. Open the specified directory and traverse it in depth-first order, building a vector of all files contained therein. Each element in the vector is a `string` specifiying the path of the file relative to your working directory. At this point, you do not know whether these files contain an image. Use the following methods to build the vector:

   Apple/Linux Use the `dirent` functions to open and read the directory entries. However, the function to check for directory in Linux (comparing the field `d_type` to `DT_DIR`) is finicky. It depends on the underlying file system and is know to work with `ext3` or `ext4` but is not guaranteed to work with other file systems. Therefore, I recommend that you use `lstat` to get metadata on file and use the macro `S_ISDIR` to determine if the file is a directory.

   Windows Download the file `dirent.h` and use the standard functions to open and read directory entries. You can use the comparison of `d_type` field to `DT_DIR` to determine if the file is a directory. You can use Google to search for `dirent.h` on the web.

3. For each file in your vector of files, read the file contents as an image (using `cv::imread`). If it is an image, display it and wait for user response. If it is not an image, delete the entry from your vector and move on to the next entry. The valid inputs from user are: space bar or `n` (for next image), `p` (for previous image), and `q` (to stop the program). Display the image information (name, size) in the console window.

4. Most of the images produced by the current generation of digital cameras are too big to fit onscreen. For example, a 22 megapixel camera produces an image of size 5760×3840 pixels while a large screen is barely 1920×1080 pixels. Your screen may be smaller and the default display window dimensions are also small. So, you want to make sure that the image completely fits in the display window. You can achieve that by using the affine transformation function of OpenCV that will preserve the aspect ratio. Look for the functions `cv::getAffineTransform` and `cv::warpAffine`.

5. Make sure that you catch any exceptions thrown by OpenCV functions. They will help to save you a lot of trouble.

Criteria for Success

You are free to use your own methods to solve the problem. For example, you can use X11 library of functions to determine the window resolution in Linux or Apple but be mindful of the fact that the person compiling your code may not have proper header files on their machine. In such a case, provide clear instructions on what is needed in the file `README`. You are also free to device your own mechanism to build a data structure for all the files, instead of a vector of strings as suggested.

The GUI interface should respond to the following keys when the focus is on image display window:

space or `n` Display next image

`p` Display previous image

`q` Quit the program

Grading

I'll use the following rubric to assess your submission.

1. *Overall submission; 30pts* Program compiles and upon reading, seems to be able to solve the assigned problem.

2. *Command line parsing; 10pts* Program is able to parse the command line appropriately, assigning defaults as needed; issues help if needed.

3. *Accessing images; 10pts* Program is able to access all images in depth-first order; displays all images and ignores the files that are not recognized as images; traverses directories as needed to an arbitrary depth

4. *Conformance to specification; 20pts* Program works correctly and meets all of the specifications, including display, navigation, image metadata display, and image resizing.

5. *Code readability; 10pts* The code must be readable, with appropriate comments. Author and date should be identified.
6. *Code reusability; 10pts* The code is structured in the form of functions, in separate files with their own headers, to facilitate reusability.
7. *Exception handling; 5pts* The code should handle exceptions, thrown from within the code or OpenCV.
8. *Meta-files, 5pts* Meta-files such as `README`, `Makefile`, and revision history are provided and are appropriately written.

I'll really like if you use `doxygen` to generate user documentation for your code though I do not require you to do so.

Submission

Submit an electronic copy of all the sources, `README`, Makefile(s), and results. Create your programs in a directory called *username*.1 where *username* is your login name on delmar. This directory should be located in your `$HOME`. Once you are done with everything, *remove the executables and object files*, and issue the following commands:

```
% cd
% chmod 755 ~
% ~bhatias/bin/handin cs5420 1
% chmod 700 ~
```

Do not copy-paste these commands from the PDF; type in those commands.