Assignment DS2

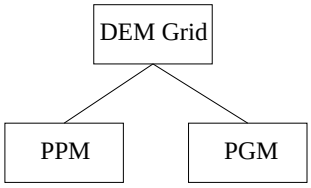
Due Date: February 18

Purpose

This project brings together arrays, classes, methods, and an inheritance hierarchy. You will also use real-world data to create some interesting map visualizations.

Problem

GPS, Google Earth, and other applications display terrain, using different colors to represent elevation. However, sometimes data is missing, and the terrain must be interpolated before it can be viewed. Furthermore, such visualizations can use various file formats. In this project, you will create colorful maps and interpolate areas where the data is incomplete. You will read a common data file format for elevation values: DEM ASCII grid format. This human-readable format stores a grid of floating point elevation values by rows. See "Esri grid" on Wikipedia for more info. You will use two different data file formats to store and visualize real terrain (PGM and PPM formats; see Wikipedia). Data from these file formats will be stored in an inheritance hierarchy, shown below:



Input

Your program should repeatedly offer the user a menu similar to the following:

```
Menu ~~~~
cpgm - Create new PGM object
cppm - Create new PPM object
spgm - Display number of rows and columns of the current PGM instance
sppm - Display number of rows and columns of the current PPM instance
hpgm - Display highest elevation of the current PGM instance
lppm - Display lowest elevation of the current PPM instance
wpgm - Write current PGM to output file
wppm - Write current PPM to output file
rmse - Display the RMSE of the two current instances
quit - Quit
```

Elaboration:

- `cpgm` - prompt the user for a filename in DEM ASCII grid format (.grd) and create a Portable GrayMap (PGM) object. First the data for the DEM should be read, stored, and interpolated (if needed). Then a PGM should be computed in gray scale, from 50 (lowest elevation) to 255 (highest elevation).
- `cppm` - prompt the user for a filename in DEM ASCII grid format (.grd) and create a Portable PixMap (PPM) object. First the data for the DEM should be read, stored, and interpolated (if needed). Then a PPM should be computed in color, following a green to yellow gradient; that is, the lowest elevations should be in dark green, moving to very light green in highest elevations.

- `spgm`|`sppm` - display the number of rows/columns. Do this in an orderly, aligned way with clear labels.
- `hpgm` - display the highest elevation in an orderly, aligned way with a clear label.
- `lppm` - display the lowest elevation in an orderly, aligned way with a clear label.
- `wpgm`|`wppm` - prompt for the output file name and write the appropriate PGM (.pgm) or PPM (.ppm) file to that file.
- `rmse` - compute the root mean square error (RMSE) of the current instances of PPM and PGM objects. If none or only one exists, a message to that effect should be displayed. The RMSE is computed as follows:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(u_i - w_i)^2} \tag{1}$$

where u_i = the interpolated DEM elevation of test point i
w_i = the true (most probable) elevation of test point i
N = the total number of elevation points

What RMSE does is compare two elevation files to determine their differences. If the two files are identical, then the RMSE = 0. Generally, we use RMSE to compare two DEMs of the same quadrangle to see how well they match up. The smaller the RMSE, the better the match. None of that really matters here; just display the RMSE to two decimal places.

You may assume the user will input the correct data; no error checking is needed, except for the input menu option and opening a file. For this former, an error message should be displayed and the program should continue, redisplay the menu, and offer the user another chance to input a valid option. For the latter, the program should check that a file is valid before trying to read the contents. If a file is not valid, then an error message should be displayed and the menu redisplayed, as above.

Careful thinking about this shows that you can have two different file formats active at the same time, but only **one** of each; that is, the user can create a PGM from one set of data and another PPM from another set of data, but if the user then creates another PPM, for example, the previous one is replaced by the new data.

Output

The program should display the menu and input messages, as appropriate. The program should write the desired files when the 'w' option is chosen. The user then has to look at the output using an appropriate image viewer on their particular computer/operating system.

Below is an example of a PGM of Mt. Washington, NH, created from the DEM ASCII grid file `tucks.grd`.

Specifics

- You must use the hierarchy as described above.
- Only those things specific to a class should be done by methods in that class. Reuse code from the base class as much as possible.
- The algorithm for how to interpolate data will be discussed in class.
- The number and complexity of methods is up to you. But as a general rule, a method should be relatively short – less then approximately 16 lines. If a method gets longer than that, you should probably split the problem into several methods.
- Include the usual comments as was done in the previous project.
- The base class should process **only** DEMs in grid format. The base class does now "know" about PGM or PPM files, so do not include color or other information specific to the derived classes in the base class.

Notes

Don't wait to start work on this. Projects take longer to complete than you might realize. As always, work on one aspect at a time to minimize the number of problems you encounter.

You should test with multiple data sets with different "missing" values. Don't start testing with the large files available online; create some small files that allow you to debug more easily.

Be sure your program does not crash when **valid** input is entered.

To submit, create a header file (`.h`) file that contains all of your class specifications and your introductory comment. Create an implementation (`.cpp`) file that contains all of the methods. Finally, you should have a third `.cpp` file that contains your `main()`. Zip all of this together; name this file *first_initial+last_name*`DS2.zip`, as in `mgousieDS2.zip`. Submit the file through Canvas before the due date. Submit hard copy of your code in class the next day. Write (or type) and **sign** the Honor Code on the hard copy.

Climb the mountains and get their good tidings.
Nature's peace will flow into you as sunshine into trees.
– John Muir