# HW2

Please complete the coding exercises below, and submit a .zip file of your source code on D2L.

---

### Q1. Class construction (7 points)

During the 'Pointers' lecture, we created and used a 'float **xtable' 2D array, like so:

```
int xRes=512, yRes=256;

float **xtable; // pointer to pointer[s]
xtable = new float*[yRes]; // a ("column") array to hold 'float *' type pointers

// each element of the above array points to a 'row' of xRes elements
for(int i=0;i < yRes;i++) {
  xtable[i] = new float[xRes];
}

for(int i=0;i < yRes;i++){
   for(int j=0;j < xRes;j++){
       xtable[i][j]=45; // store 45 for pixel data, "for now"
   }
}
```

**Create an 'array2D' class that encapsulates the above.** Your class should contain the following 'protected' members ["protected members are accessible in the class that defines them, AND in classes that inherit from it"]:

- xtable
- xRes
- yRes

It should also contain the following public interface:

- constructor: array2D(xResolution,yResolution)
- getSize(int &xResolution, int &yResolution) ['&' because we want those vars to be SET inside the function, like a 'return' in a way; confusingly, int xResolution&, int yResolution& means the same thing!]
- setValue(x,y,val) [at (x,y), store val (in xtable, y would be the rowID, x, the columnID)]
- val=getValue(x,y) [at (x,y), retrieve val]

Remember to create a destructor function as well :)

Exercise your array2D class as follows, inside main():

```
array2D *a = new array2D(320,240);
int xRes, yRes;
a->getSize(xRes,yRes);

//NOTE the j,i ordering below (twice), since we specify coords as (x,y) [not (y,x)]
for(int i=0;i < yRes;i++){
   for(int j=0;j < xRes;j++){
       a->setValue(j,i,100); // constant value of 100 at all locations
   }
}
for(int i=0;i < yRes;i++){
   for(int j=0;j < xRes;j++){
       cout << a->getValue(j,i) << " ";
   }
   cout << endl;
}

delete a;
```

All the code needs to be in a single Q1.cpp file.

---

**Q2. Class inheritance (8 points)**

**Extend (inherit from) the array2D class from Q1 above, to create a 'PGMImage' class.**

```
class PGMImage: public array2D {
   // private:
   // ...
   // protected:
   // ...
   // public:
   // ...
};//PGMImage
```

PGMImage 'inherits' [gets handed down] all the public and protected members and methods of array2D.

PGMImage should contain this additional non-public (private or protected) member:

• string filename (or char filename[2048])

It should contain the following public methods:

• PGMImage(xResolution,yResolution,imageFilename)

• getResolution(xResolution&,yResolution&)

• setPixel(x,y,val) [at (x,y), store val]

• val=getPixel(x,y) [at (x,y), retrieve val]

• writeFile()

In main(), write code similar to that for Q1, with 'test.pgm' as the output filename. Your code would output a test.pgm file, which should be loadable via the netpbm viewer (https://cs455-usc.updog.co/m18_CsML00Agi/extras/ImgProc/PGMViewer/viewer.html), Irfanview, etc.

Place BOTH class declarations (array2D, PGMImage) in Q2.h, #include "Q2.h" in Q2.cpp, and in Q2.cpp, provide both class definitions plus main().

---

**Q3. Class construction and image synthesis (10 points)**

**Create a ComplexNumber class, and use it, along with the PGMImage class from Q2, to create a grayscale Mandelbrot fractal image (.pgm) file.**

The ComplexNumber class would have these private members:

• double real

• double imag

It would need to implement these public methods:

• ComplexNumber(i,j)

• ComplexNumber add(ComplexNumber c2) [outputs a sum, as a new complex number]

• ComplexNumber squared() [outputs a squared value, as a new complex number]. As you might know, the real part would be real^2 - imag^2, and the imag part would be 2*real*imag. However, you can't just do 'real=(pow(real,2)-pow(imag,2)); imag=(2*real*imag);' [because real gets overwritten in the first statement, and therefore makes it incorrect for imag= in the second!].

• double abs() [outputs the magnitude of the complex number: sqrt(real*real+imag*imag)]

• void prt() [does cout in the form of 'real + imag j']

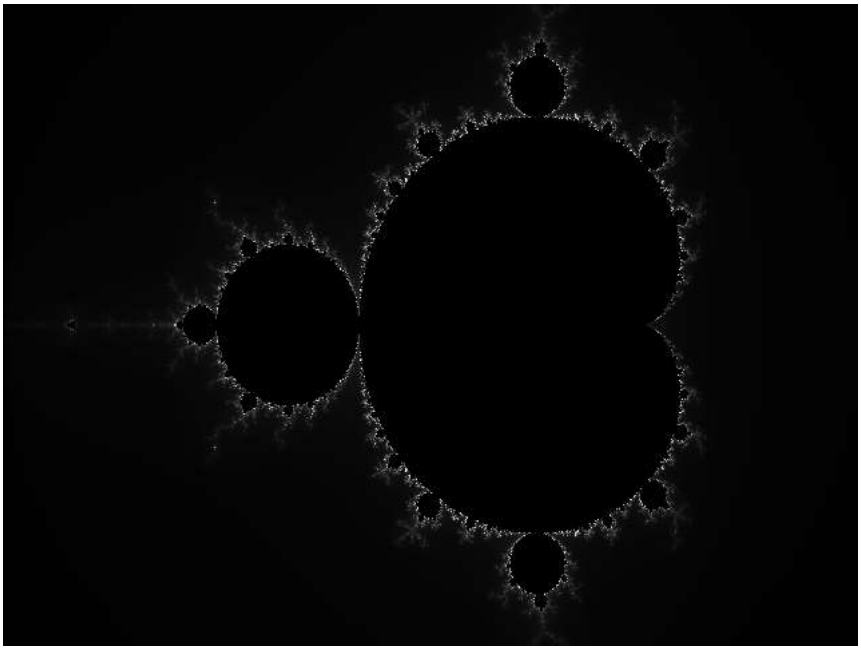
Once you create ComplexNumber, you can do things like:

```
ComplexNumber c1(3,4), c2(-2,1);
ComplexNumber c3 = c1.squared(), c4 = c3.add(c2);
```

Next, create a 'fractal.pgm' Mandelbrot fractal image file, 640x480 resolution. Here is the pseudocode:

```
create a blank image
for each y (in yRes)
  for each x (in xRes)
    x,y is our pixel coordinate. TRANSFORM this to a complex number 'c' as follows:
    c.real = cxmin + x/(xRes-1)*(cxmax-cxmin) [maps x to cxmin..cxmax]
    c.imag = cymin + y/(yRes-1)*(cymax-cymin) [maps y to cymin..cymax]
    [(cxmin,cymin) and (cxmax,cymax) are diagonal endpoints of a complex-number rectangle,
    you can take them to be (-2,-1) and (1,1) respectively]
    set z = 0
    set maxIter = 255
    set iter=0
    while (iter < maxIter and abs(z) < 2)
      z = z^2 + c [iterate!!] [use .squared() and .add() from your ComplexNumber class]
      iter++
    end_while: either iter has exceeded maxIter, or abs(z) has exceeded 2
    if abs(z)>=2 (we escaped our containing circle of radius 2!)
      set pixel value to be iter
    else (we got stuck, ie iter>maxIter but we're still inside the circle)
      set pixel value to be black
write out image
```

The above should produce the iconic Mandelbrot (https://www.google.com/search?biw=1360&bih=613&tbm=isch&sa=1&q=Mandelbrot++fractal&oq=Mandelbrot++fractal&gs_l=img.3..0l7.738 'buddha' (zoomed out) fractal image, like so [cool!]:



Place all three class declarations in Q3.h, include it in Q3.cpp, and in Q3.cpp, provide all three class' definitions and main() [main() is where the fractal image calculation should happen].

---

**Extra credit item #1, 2 points**: look up a **different (cxmin,cymin),(cxmax,cymax)** bounding box (with SMALLER values than (-2,-1) and (1,1)), and compute a different fractal - this is how very pretty zooms are created :) Eg. this page lists 9 different (cxmin,cymin) pairs [you need to add a small step, eg. 0.001, to (cxmin,cymin) to get (cxmax,cymax)]: http://www.nahee.com/Derbyshire/manguide.html (http://www.nahee.com/Derbyshire/manguide.html) You can Google search for similar pages that list more such 'interesting' areas to explore. For this extra credit, you need to pick just one new region, and use it to create one new image that is different from the (-2,-1)..(1,1) one shown above.

---

**Extra credit item #2, 3 points**: see if you can create a series of output images (eg. 24 images) where 'c' is systematically being varied (eg. to represent zooming-in, or panning), and **create a small animation** out of the resulting image sequence (using any software you want, to generate a movie file out of the still images). Any movie format is fine: .qt, .avi, .mpg, .flv, .ogv, .mp4.. You can use avidemux (http://fixounet.free.fr/avidemux/download.html (http://fixounet.free.fr/avidemux/download.html)), ImageJ (http://www.andrewnoske.com/wiki/ImageJ (http://www.andrewnoske.com/wiki/ImageJ)), etc. to convert an image sequence into a movie clip :) You can convert the .pgm images into .jpg, .png etc., using Irfanview (you would open (read in) a .pgm, and export (save as) a .png or .jpg, for each image in your sequence. Reason - many video creation programs accept .png or .jpg still images, but not .pgm :( Also, instead of a movie, you can also create (http://gifmaker.me/) and submit an animated GIF file, eg. 'fractal_zoom.gif' (eg. here (https://giphy.com/gifs/animation-fractal-mandelbrot-vZ5WKoNA4V7fa) is one, in color :)).

For the extra credit problems, the image does not need to be 640x480 - rather, best if it matches the aspect ratio of the zoom window. In other words, if (cxmax-cxmin)/(cymax-cymin) is 1.8 for example, your resolution would be 1800x1000 (since 1800/1000 is also 1.8). For this example, the resolution could also be 3600x2000, 900x500 etc. This is so the images don't appear to be stretched/squashed.

All the code should be in Q3EC1.h,Q3EC1.cpp for extra credit #1, likewise another .h,.cpp pair for #2 (separating them like that makes it easier to grade).

For all three questions plus extra credit ones, feel free to include README(.txt) files that contain FYIs, special instructions, etc.

FYI, here's the fractal generation code, in 92 different (!!!) programming languages (do NOT stare at the C or C++ one for too long!): http://rosettacode.org/wiki/Mandelbrot_set (http://rosettacode.org/wiki/Mandelbrot_set)

Please meet with any of us teaching staff, if you need clarification, additional info on something, etc.

**HAVE FUN**!!! If you get started early, it WILL be (a lot of fun)..