Due Friday, Nov 8, 11.59pm, Late Deadline, Nov 10, 11.59pm

In this project, you will develop a templated class for representing **Sets of different types**, following the book's implementation of a Set templated class that uses linked lists (Chapter 17, pages 793-798). Instead you will use an **array** to represent the set. Also the partially ordered array implementation in Chapter 16 will be useful.

Set Representation

Your Set class should hold the following attributes and implement the associated methods (may include additional attributes/methods as needed).

```
template<class T>
class Set {
   private:

      int capacity; // allocated size
      int used;     // current used size
                    // pointer to the dynamically allocated set (array)
      T *set_data;
                // pointer to item counts
      int *item_counts;
                // search  for a particular item
      bool search (T item);

   public:
      Set();      // constructor - creates a set with a default size
      Set(int cap) // constructor - creates a set with a specific capacity
      ~Set();     // destructor  - provide as many destructors as needed

                  // accessors/mutators
      int getCapacity();
              // if capacity is larger, then reallocate, copy
              // and destroy old set
      void setCapacity(int w);
              // returns the size of the set
      int getSize();

              // checks to see if an element exists in the set
      bool contains(T target);
                  // set/get an image pixel given row and column addresses
                  // pixel is a 3 element r,g,b triple
      void addElement(T item);
};
```

Note that the above representation is incomplete; a typical Set class will contain operations for union, intersection, difference, etc.

Tasks.

You will test your set implementation on multiple images, includng the Yosemite image from the previous project. Two different set implementations will be used to characterize the content of the images, (1) histogram of the most frequently occurring pixels(r,g,b triplets), and the (R,G,B) content of the whole image.

1. Implement the functions of the class; Follow the logic of the book's implementation, the main difference being the set representation - we use an array. Secondly, as items are inserted into the set, also increment

the item's count in the corresponding entry of the item_cnt array. There will be two versions of the set:

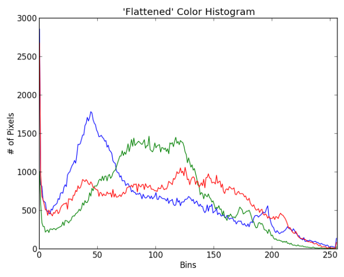
a) The set will store each item as an R, G, B triplet (this can be contiguous in the array representation). Each unique R,G,B triplet will have a count (number of occurrences) as the image is read in (stored in the item_counts array).

b) The set will store individual values of red, green and blue; in this case we are counting the individual occurrences of teh 3 components. Thus the set will have a maximum of 256*3 = 768 values (each color component is 8 bits), not all of which might occur. This can be represented as 3 separate sets, or in 1 set, with R, G, B stored sequentially in a single set (with first 256 possible values for red, followed by green and blue). You can overload the addElement() method for this purpose.

2. Once the set is created, we want to characterize the content of the image. We will do this in two ways:

   a) Identify the most common pixels in the image. You can do this by thresholding the pixels that occur above a certain count. Play with this threshold (threshold of 0 will select all pixels). Once the threshold is set, then read in the image again (unless you have it in memory), compare each pixel to the subset of 'common pixels' and mark those in some unique color (say white). The resulting image will display the most commonly occurring pixels by color.

   b) In the second case, we only have red, green and blue components for the whole image. In this case, we will plot the histogram of the image. An example is shown below:



   You may use any plotting program for this purpose, eg., gnuplot. It is useful to color the curves in the respective colors, for obvious reasons. Alternately you may draw bars for each value. Scripts for gnuplots will be provided for generating the graphs.

3. **Testing.** Images for testing will be provided. The yosemite image from prior assignments will be one of them.

4. **Output.** For each test image, output the corresponding image with the common pixels marked (use a suitable threshold, but should be user specified on console) and the histogram data.

5. **Documentation.** Generate an updated doxygen documentation of your sources; **each of your methods should be documented, including each input parameter, return value, etc.**

Evaluation:

- As per rubric (on Canvas).
- **To Turn in to Canvas:** All source code files, output from your implementation, doxygen based documentation.
- **Late Policy.** Upto 2 days and for a maximum of 75% credit. No credit 2 days beyond the deadline.