4 The point set class

The next and the last class you need in the project is the point set class: PointSet. Each element in a point set is a 2-D point. The implementation of a point set is array-based. So a set is stored in an array, where each item in the array is a point. The header file for the point set class, PointSet.h, can be downloaded from the project page. As before, you can not alter the given header file. To complete this part of the project, you need to implement the point set class except for `qsort`, `closestPointBF`, `closestPointDC`, `quickSort`, and `closestPoints`, which will be the tasks for the next step of the project. (You may leave the implementation of these functions empty for the time being.) To receive credit for this part of the project, submit your `PointSet.cpp` by the midnight of the due date.

5 Two ways to get input data

To test your program, you will need some input data (point sets). Of course you can provide the input data via keyboard. However, it is time-consuming especially when a point set you wish to use contains hundreds or thousands of points. There are two better and more efficient ways to get input data. The first method is to read a point set from an input file. Each line in the file contains the *x* and *y*-coordinates of one point in the form of two positive integers separated by a space. To test the program using this input option, you should first create such a file of points. The second method to get input data is to use a random number generator, which generates a number randomly in a pre-specified range. There are many good random number generators. However, the easiest way to generate a random number is to use the system time as the seed and the system call *rand* to get a random number. The following is a simple program that generates 20 random numbers in the range of 0 and 99.

```
#include <stdio.h>
#include <sys/time.h>
#include <stdlib.h>

int main (void)
{
  int i, j;
  struct timeval tp;
  gettimeofday (&tp, 0);
  srand (tp.tv_sec + tp.tv_usec);
  for (i = 0; i < 20; i++) {
    j = rand () % 100;
    printf ("random number %d\n", j);
  }
  return 0;
}
```

6 Memory implementation of a point set

As indicated in the header file, a point set is stored in an array. The default array size is 256, which is set in the constructor implementation. However, before an operation that will increase the size of a point set, such as adding a point to the set, your program must compare the current set size with the array size to decide if it is necessary to resize the array. This is when the protected function `resize()` is needed. It will first double the array size, declare a new array of the increased size, then copy the current (full) array to the new array, and finally delete the old array. The memory implementation is a very important part of the project since it directly affects the implementations of other functions in the class. Make sure the memory allocation works correctly before you go on to the other functions.

To be continued...