ECS 60 **Programming Assignment #5, 50 points (2.5 hours)** **Winter 2017**

Due: Thursday, March 16th at 11:59pm to p5 directory
Minimum files to submit: authors.csv, Makefile, router.cpp, and router.h
Executable name: router.out Makefile name: Makefile

For this assignment, you will be writing a program determine the paths to connect all of the cities on a map.
You are to handin all files upon which your program is dependent, except RouterRunner.cpp, RouterRunner.h, and CPUTimer.h. The grading script will copy those three files into your directory. You will find those three files, map files, CreateFile.out, ElevationDiffs.out, barebones router.*, minimal Makefile, and my router.out. in ~ssdavis/60/p5.

Here are the specifications:
1. CreateFile.cpp is available and worth perusing.
   1.1. The name of a data file reflects parameters used to create it. For example, map-20-1000-5.txt has 20 cities, a map width and height of 1000, and was created using a seed of 5 for the random number generator.
   1.2. The first line of the file indicates the number of cities, and the width. All maps are square.
   1.3. The next lines provide the coordinates of the cities. The origin of the coordinate system is the lower left corner.
   1.4. The rest of the lines are the elevations of each plot in the map, starting from the upper left and working across, and then down. Thus, the map in the file matches the map stored with respect to its origin.
   1.5. ElevationDiffs.out asks for a map filename, and then counts the number of differences in height between all of the adjacent map plots, and provides them in a list of those counts for differences from 0 to 99.
2. General Operation
   2.1. main() calls readFile() to read the file into two two-dimensional maps, as well as an array of the cities.
   2.2. The CPUTimer is then started.
   2.3. One of the plot maps is passed to the Router constructor. When the constructor returns, that map is destroyed, so you must make some copy in your constructor.
   2.4. `void Router::findRoutes(const CityPos *cityPos, int cityCount, Edge *paths, int &pathCount)` is the real workhorse of the program. This function must place Edges in paths sufficient to connect all of the cities, and set pathCount to the number of Edges used in the paths array.
      2.4.1. Paths may split at intermediate points between cities!
   2.5. When findRoutes() returns, the CPUTime is printed.
   2.6. checkRoute() ensures that all submitted Edges are for adjacent plots. It also uses dfs to determine the total cost, and to ensure that all cities are connected to each other. If it detects an error, it posts a message indicating the problem, and exits.
3. Grading
   3.1. Performance will be tested with three map files, each with twenty cities, and a width of 1000.
   3.2. (20 points) Correctly connect all of the cities. This is indicated by having no messages from checkRoute(). If a program does not correctly connect the cities, then it will receive zero for the entire assignment.
   3.3. (15 points) CPU time: min (18, 15 * Sean's CPU Time / Your CPU Time);
      3.3.1. CPU time may not exceed 60.
      3.3.2. Programs must be compiled without any optimization options. You may not use any precompiled code, including the STL and assembly.
   3.4. (15 points) Total cost of the paths in the paths array: min(17, 15 * Sean's total / Your total)
      3.4.1. Cost to connect adjacent plots = $(\text{change in elevation})^2 + 10$.
      3.4.2. Each non-edge plot has eight adjacent plots.
4. Suggestions
   4.1. Keep things simple, and get things running first. After it works, then refine it. I wrote mine in 2 hours, with minimal debugging needed. I then spent less than 30 minutes making a single tweak that reduced CPU time in half. Ignoring the Weiss code that I modified, my router.* are only 243 lines.
   4.2. For debugging, test in this order: map-2-5-2.txt, map-3-5-1.txt, map-5-10-3.txt, map-3-100-4.txt, map-5-100-6.txt, map-20-1000-7.txt, map-20-1000-8.txt, map-20-1000-9.txt.
   4.3. Use Weiss code where possible, but modify it to eliminate obvious slow downs.
   4.4. Remember to turn in dsexceptions.h if your program needs it!

```
typedef struct{
  int x;
  int y;
} CityPos;

typedef struct {  // start and end should be adjacent!
  short startX;
  short startY;
  short endX;
  short endY;
} Edge;

typedef struct {
  short map1000[1000][1000];
} Map1000;

int main(int argc, char* argv[])
{
  Map1000 *map, *map2;
  int width, cityCount, pathCount;
  CityPos *cityPos;
  Router *router;
  Edge *paths;
  CPUTimer ct;
  map = new Map1000;
  map2 = new Map1000;
  readFile(argv[1], map, map2, &width, &cityPos, &cityCount);
  paths = new Edge[4 * width * width];  // maximum number of edges possible
  ct.reset();
  router = new Router((const Map1000*) map, width);
  delete map;
  router->findRoutes((const CityPos*) cityPos, cityCount, paths, pathCount);
  cout << "CPU time: " << ct.cur_CPUTime() ;
  checkRoute(map2, cityPos, cityCount, paths, pathCount, width);
  delete map2;
  delete [] cityPos;
  delete [] paths;
  delete router;
  return 0;
}
```

```
[ssdavis@lect1 p5]$ cat map-3-5-1.txt
3 5
1 3
0 2
0 3
  3 55 29  1  2
  5 62 46  3  6
  3 59 83  4  8
  5 72 51  3  6
  4 71 34  2  6
[ssdavis@lect1 p5]$ router.out map-3-5-1.txt
CPU time: 0.006986 Total cost: 3505
[ssdavis@lect1 p5]$ router.out map-20-1000-7.txt
CPU time: 0.415207 Total cost: 70548
[ssdavis@lect1 p5]$ router.out map-20-1000-8.txt
CPU time: 0.393765 Total cost: 69622
[ssdavis@lect1 p5]$ router.out map-20-1000-9.txt
CPU time: 0.443063 Total cost: 78411
[ssdavis@lect1 p5]$
```