Due Tuesday, August 1st, 11:59pm.  
Executable name: p5.out

Files NOT to handin to cs60 p5 are: main.cpp, CPUTimer.h

Minimum files to handin to cs60 p5 are: authors.csv, Makefile, connector.h, connector.cpp.

You are to write an efficient program that will find the minimum spanning tree for all the cities on a map. I have provided the driver program main.cpp, and the necessary class stubs in connector.h and connector.cpp. You may add any classes and/or code you wish to connector.cpp, and connector.h. You may also use, and alter any Weiss files. You may also use any other C++ files of your own making. Just remember to handin all of the files necessary to compile your program, except the three files noted above. All files mentioned, as well as my executable, p5.out, can be found in ~ssdavis/60/p5

Further specifications:
1. Command Line parameter is the name of the data file, followed by either a ‘0’ or ‘1’. A ‘0’ indicates that the program should not show the connections selected.

2. Data Files
   2.1. Since main.cpp handles all file input, you need not concerned with how to read the data files.
   2.2. The files are created by CreateMap.out which is compiled from CreateMap.cpp. These are available in p5.
   2.3. Data file names are appended with two numbers: the length of a side of a map, and the seed used for the random number generator.
   2.4. The first line of data files is the length of the side.
   2.5. After the first line are |side| binary lines of the map. The range of elevations vary from 1 to 255, and are logarithmically distributed starting from 0.
   2.6. Following the map lines is a binary listing of the coordinates of the cities. There are |side| cities. Their coordinates are guaranteed to be unique.

3. Connections are a set of four integers, y1, x1, y2, x2, where x1 and y1 specify the coordinates of map location that is adjacent to y2, x2.
   3.1. There are eight adjacent coordinates to positions in the middle of the map.
   3.2. The cost of a connection is abs(map[y1][x1] – map[y2][x2]).
   3.3. Your program fills the connections[][4] array and then sets numConnections to the number of connections in the array.
   3.4. Only city locations may have more than two connections.

4. Measurements
   4.1. CPU time starts just before constructing your Connector object in main(), and ends after the call to connect(). Thus, your destructors will not be called during CPU time.
      4.1.1. You may not have any global variables since they would be constructed before CPU time starts.
      4.1.2. Measurements will be made with the ‘0’ command line option so that screen output will not slow your program.
   4.2. Grand Total is the total of all the connection costs. Your Grand Total must be no more than twice mine, or the program will receive zero.

5. Grading
   5.1. The program will be tested using three data files each with side of length 500. The measurements will be the total of the three runs.
   5.2. If there are ANY error messages from checker within main.cpp, then the program will receive zero. If the program prints nothing but the CPU time and Cost total, then your connections have no detectable errors.
   5.3. CPU Time score = min(30, 25 * Sean's CPU / Your CPU)
   5.4. Cost score = min(28, 25 * Sean’s Cost / Your Cost).
   5.5. I reserve the right to set a maximum CPU time and a maximum Cost for programs.

6. Makefile. The Makefile provided contains the minimum needed to create p5.out.
   6.1. You may not use an optimization flags in your Makefile.
   6.2. All code must be C++ source code.
   6.3. To ensure that you handin all of the files necessary for your program, you should create a temp directory, and copy only *.cpp, *.h, and Makefile into it. Then try to make the program. Many students have forgotten to handin dsexceptions.h.

7. Suggestions.
   7.1. Start early, and get something working without errors.
7.2. Don’t fuss about speed or size until you have something working. Too many students fail to have a correct program by the deadline because they spend too much time tweaking early on.

7.2.1. You will learn a lot just getting it running. Then you can use this knowledge to improve your code.

7.3. You can create your own small data files using CreateMap.out.

7.4. You can look at a map using the ShowMap.out in p5. A ‘C’ before an elevation indicates a city.

7.5. Leave lots of time for testing and debugging. Test with all of the available files.

```c
#include <stdlib.h>
#include <string.h>

int main(int argc, char* argv[]) {
    const unsigned char **map;
    short cities[1000][2], **connections;
    int side, numConnections, cost;
    CPUTimer ct;
    map = (const unsigned char**) readFile(&side, argv[1], cities);
    connections = new short* [200 * side];
    for(int i = 0; i < 200 * side; i++)
        connections[i] = new short[4];
    ct.reset();
    Connector *connector = new Connector(side);
    cost = connector->connect(map, side, cities, connections, numConnections);
    cout << "CPU Time: " << ct.cur_CPUTime() << " Cost: " << cost << endl;
    if(argv[2][0] != '0')
        for(int i = 0; i < numConnections; i++)
            for(int j = 0; j < 4; j++)
                cout << connections[i][j] << " ";
    cout << endl;
    checkConnections(map, side, cities, connections, numConnections);
    checkCost(map, side, cities, connections, numConnections, cost);
    return 0;
}
```

```
[ssdavis@lect1 p5]$ ShowMap.out Map5-5.txt
0 0 1 2 3 4
0: C21  f  d  c  7
1: 11 12 25 8  5
2:  f 26  bc 9  8
3:  c  c  8C 8C 6
4: 8 bc 8 8 6
Your choice (d, D, r, R, l, L, p, u, U, x) : x
[ssdavis@lect1 p5]$ p5.out Map5-5.txt
CPU Time: 0 Cost: 29
```

```
[ssdavis@lect1 p5]$ p5.out Map500-7.txt
0 0 1 2 3 4
CPU Time: 0.741543 Cost: 63305
[ssdavis@lect1 p5]$ p5.out Map500-8.txt
CPU Time: 0.757742 Cost: 66163
[ssdavis@lect1 p5]$ p5.out Map500-9.txt
CPU Time: 0.41805 Cost: 49467
```

```
[ssdavis@lect1 p5]$
```