In this programming project, you will be implementing Dijkstra's shortest path algorithm in a directed edge weighted graph. You should use the C++ programming language, not any other programming language. Your program should be based on the g++ compiler on general.asu.edu. **All programs will be compiled and graded on general.asu.edu, a Linux based machine**. If you program does not work on that machine, you will receive no credit for this assignment. You will need to submit it electronically at the blackboard, in one zip file, named CSE310-P03-Lname-Fname, where Lname is your last name and Fname is your first name. The zip file should contain a set of files that are absolutely necessary to compile and execute your program. If you program does not compile on general.asu.edu, you will receive 0 on this project.

Your program should take one of the following four commands from the standard input, and execute corresponding functions.

- **S**
- **R**
- **W**
- **P s t**

On reading **S**, the program stops.
On reading **R**, the program reads in an edge weighted directed graph from file GRAPHinput.txt to build the adjacency lists, and waits for the next command.

The file GRAPHinput.txt is a text file. The first line of the file contains two integers n and m, which indicates the number of vertices and the number of edges on the graph, respectively. It is followed by m lines, where each line contains three integers: u, v, and w. These three integers indicate the information of an edge: there is an edge pointing from u to v, with weight w. Please note that the vertices of the graph are indexed from 1 to n (not from 0 to $n-1$).
On reading **W**, the program writes the graph information to the screen, and waits for the next command.

The screen output format of W is as follows: The first line should contain two integers, n and m, where n is the number of vertices and m is the number of edges. It should be followed by n lines, where each of these n lines has the following format:

$$u\ : (v_1\ w_1)\ (v_2\ w_2)\ ...\ (v_x,\ w_x)$$

On reading **P `s` `t`**, the program runs Dijkstra's shortest path algorithm to compute a shortest *s*-*t* path, and prints out the shortest path information from *s* to *t* to the screen, and waits for the next command.

The screen output format of `P s t` is as follows: There are two lines. The first line contains an integer *dist*, which is the length of the shortest path computed. The next line contains the indexes of all vertices along the path direction, starting from *src* and ending with *dest*, in format of:

$$s\ v_1\ v_2\ ...\ t$$

Please note that your program should read in only one graph, but may be asked to compute *s*-*t* shortest paths for different pairs of *s* and *t*. Therefore, during the computation of the *s*-*t* shortest path, your program should not modify the given graph.

You should use modular design. At the minimum, you should have

- the main program as `main.cpp` and the corresponding `main.h`;
- the heap functions `heap.cpp` and the corresponding `heap.h`;
- the graph functions `graph.cpp` and the corresponding `graph.h`;
- various utility functions `util.cpp` and the corresponding `util.h`.

You should also provide a `Makefile` which compile the files into an executable file named `run`.

Grading policies: (Sample test cases will be posted soon.)

(10 pts) Documentation: You should provide sufficient comment about the variables and algorithms.

(10 pts) Makefile

(10 pts) Modular Design

(10 pts) Graph I/O

(20 pts) Shortest Path Length

(20 pts) Shortest Path Path

Above all, you need to write a working program to correctly parse the commands specified in the project. Without this, your program will not be graded.