1. The following is a pseudo-code description of a solution to the problem of finding a minimum cost network given a collection of cities and a collection of cables, where each cable directly connects two cities and has a cost. A spanning network is a subset of given cables such that any city has at least one path to any other city which runs solely through cables that are in the subset. A minimum cost network is the spanning network such that the sum of the costs of the cables in the network is minimum over all spanning networks.

1. Order the cables by increasing cost.
2. Let $N$ represent the network of cables to be output, make it empty. 
3. Repeat the following until all the cables are considered.
   3a. Choose the next cable $c$ in the ordered list.
   3b. Check whether including $c$ in the solution $N$ allows you to trace more than one path between some pair of cities using only cables already placed in $N$.
   3c. If it does, forget $c$ and continue the loop.
   3d. Otherwise, include $c$ in $N$.
4. Return $N$.

(a) In class we said that the test of step 3b. could be made simple if a group number could be assigned to each city with the meaning that two cities are connected by cables currently in $N$ if and only if they have the same group numbers. What structure did we use to record that information? What was the size of the structure? How was the structure initialized (what values did the structure hold before starting the process depicted by pseudo-code above)?

(b) Write a few lines of code that update the structure’s values when a cable is added to $N$. 

(c) In class we briefly discussed solving this problem by starting with all cables in $N$ and removing them, one at a time, if a certain test was true, until all were tested. Those remaining in $N$ would be output. Modify the update in question 1b to support a test for the connectedness of a pair of cities that is appropriate for this solution to the problem.

(d) Write pseudo-code for this solution in the style of the pseudo-code that was presented at the outset of question 1.
For homework we considered solving the problem of listing objects with dependencies from left to right where every object in the list is dependent only on objects that are to its left in the list. We called this the problem of topologically sorting a partial order. We create a class called Object. An object of this class will have an identity that is just a character string and a list of dependencies. We start writing this class as follows:

```cpp
class Object {
    char *ident; // the identity of an object
    .... depends; // a list of dependencies
    int ndepends; // the number of dependencies in the list

public:
    Object (char *str) {
        this->ident = new char[strlen(str)+1];
        strcpy(this->ident,str);
        depends = NULL;
        ndepends = 0;
    }
};
```

(a) Other than methods and the destructor, what has been left out?

(b) What would you replace .... with (you will use it below so choose it carefully)?

(c) Other than the methods that may be used to set the dependencies as they are read from file or may be used to display the dependencies there is a central method that is used to do the sort. Call this method topo. This method ensures that all objects the invoking object depends on are output (that is, in the output list) before the invoking object is. This method also tests whether a topological sort is not possible and exits if that is determined. Write a reasonable implementation of the topo method (in c++ or java) and replace the .... with the input argument.

```cpp
public void topo (...) {
```

Suppose we have an array of Object objects whose dependencies have already been set. Write c++ or java code that will perform a topological sort of the objects in the array (that is, complete the function below):

```c++
void performTopoSort (Object **objects, int nobjects) {
}
```

3. This part is just to examine your understanding of c++. If I ask what type is obj if it is declared as int obj; you would answer type int. If I ask how much space obj uses, you would answer 4 bytes. But the questions below are intended to probe a bit more than that.

(a) What type is obj1 if it is declared as int *obj1;?

(b) What type is obj2 if it is declared as int **obj2;?

(c) How much space does obj2 use?

(d) After doing this: obj2 = new int*[100];, how much space does obj2 use?

(e) What is the result if, after executing only the above lines, we execute this:
   ```
   obj2[0] = new int(45);
   ```

(f) How much space does obj3 use if it is declared like this:
   ```
   int obj3[100];
   ```

(g) If class Object is as defined as you see it in question 2 (with your replacement for . . .), and if obj4 is declared like this:
   ```
   Object **obj4 = new Object*[100];
   ```
   then how much space does obj4[0] use?

(h) Repeat the above for obj5[0] if obj5 is declared like this:
   ```
   Object **obj5;
   ```

(i) If I declare Object **obj6 = new Object*[100]; and then do this:
   ```
   obj6[0] = new Object("Hello");
   ```
   can I do this: obj6[0]->topo(...) (where . . . is what you provided above)?

(j) Sticking with obj6, can you do this: obj6[0].topo(...);?

(k) What about this: obj6[1]->topo(...);?