1. Consider the Minimum Cost Network problem. Recall the problem is to find a subset of given cables with least total cost that connects all given cities. To facilitate checking whether adding a cable to a partial solution results in a cycle we associate group numbers with each city by means of an array called group. Since a cable has its end city information immediately available, we can immediately find the group numbers of the two end cities and compare them. If the group numbers are different, the cable is added to the partial solution. But then, group numbers of cities matching the group number of one of the end cities must be changed to match the group number of the other end city. The following is code that may be used to do that:

```cpp
if (group[cable->city1] != group[cable->city2]) {
    int g1 = group[cable->city1];
    int g2 = group[cable->city2];
    for (int i=0 ; i <= max_city ; i++) if (group[i] == g2) group[i] = g1;
}
```

where `max_city` is the highest identity of any city. The problem with this solution is that group numbers of all cities need to be checked (via the for loop). This is extremely inefficient. An improved regrouping algorithm is based on the following idea. Let each City object maintain a pointer, called group, to a City object. For all cities in the same group, let one be the representative for the group. The value of group for the representative is its address. The value of group for all other cities in the same group is the address of another city in the group and values are set so that following group pointers from city to city leads to the representative city. Then, to merge the groups of two end cities of a cable one merely needs to set the value of one end city’s representative’s group variable to the address of the other end city’s representative. For example, consider the following code snippet which builds two groups, one consisting of cities 0, 1, 2 and the other of cities 3, 4:

```cpp
class City {
    public:
        City *group; int ident;
        City (int ident) { this->ident = ident; }
    };

    City **cities = new City*[ncities];
    cities[0] = new City(0);
    cities[1] = new City(1);
    cities[2] = new City(2);
    cities[3] = new City(3);
    cities[4] = new City(4);
    cities[0]->group = cities[0];
    cities[1]->group = cities[0];
    cities[2]->group = cities[0];
    cities[3]->group = cities[4];
    cities[4]->group = cities[4];
```
then, if end cities have identities 1 and 3, their representatives have identities 0 and 4 and the two groups can be merged using, for example,

```
cities[4]->group = cities[0];
```

Notice that now cities[0] refers to the city that is representative for the entire group and the representative is reachable via the group variable from all cities. The following method, added to class City, is intended to return the address of the representative of a city’s group:

```
City *representingCity() {
    if (group == this) return this;
    return group->representingCity();
}
```

Describe how this method works and list any assumptions necessary for it to work correctly, including initialization. What change should be made to the constructor of class City?

Once group representatives are found, the two groups may be merged using the following:

```
void mergeGroups(City *rep1, City *rep2) {
    rep1->group = rep2;
}
```

where should mergeGroups be put? In class City? In main? Or where? ________.

However, to reduce the time it takes to invoke representingCity, it is far better to merge a smaller group to a larger group. In other words, if rep1 represents a smaller group than rep2 then rep1->group = rep2 should be used, otherwise rep2->group = rep1 should be used. So, we add int size; to class City. This variable is intended to hold the number of cities in a group represented by a city. Modify mergeGroups to use and update the size variable. How should this variable be initialized?
Finally, it is still better to remember all City objects visited when invoking `representingCity` and then set the `group` variable of all visited City objects to the address of the new representative of the combined group. Modify `mergeGroups` and `representingCity` to implement this change.
2. The following code may show up in a List class. What is it intended to do? On what type of list is it intended to operate? What changes should be made if head is a pointer to Cell objects where Cell objects are used as carriers of listed objects (the important ones)?

```c
void what_it_do (bool (*find)(void *obj, void *what), void *item) {
    if (find(head->item, item)) {
        head = head->next;
        return;
    }
    for (Cell *ptr=head ; ptr->next != NULL ; ptr=ptr->next) {
        if (find(ptr->next->item, item)) {
            ptr->next = ptr->next->next;
            return;
        }
    }
}
```