Problem 2:

1. In addEdges, a lot of people had problems with using list_head_insert.

   the call *list_head_insert(myVertices[i].link(),name1);* will fail.
   The reason for this is that list_head_insert's first parameter is node<T>*&.
   The & means that it needs to be passed by reference, thus only variables that could actually be changed can be used.
   For example, 10 can't be passed by reference, since 10=x would be illegal. But int x, would be valid, since x=10 is legal.

   Since link() only returns a memory address(thats all a pointer is), it can't be used in the function call. However, you can make a temp variable of type node<T>* and set it equal to myVertices[i].link(), send it as the parameter, then use myVertices [i].set_link(temp) afterwards, as follows:

```
node<T>* temp;
temp=myVertices[i].link();
list_head_insert(temp,name1);
myVertices[i].set_link(temp);
```

In getNeighbors, some people had problems with the fact that getNeighbors was constant. This meant that you needed to send back a copy of the list, not the list itself. The best way to do this was to use the list_copy function.

A note on the list_copy function is that it requires 3 parameters. The first is the original linked_list's head pointer, the second will be set to be the new list's head pointer, and the third will be set to be the new linked list's tail pointer. Even if you don't care about tail, we need to send in a variable for it, since that is what it expects.

Things that are appropriate and not appropriate to be public members of your classes:

Appropriate: an accessor that returns the information needed to access the class, without making any assumptions about how the class is implemented

Inappropriate: anything that allows direct access to a private variable

anything that implies how you implemented the class
i.e. getVertex(int i) or operator[], both imply that you are implementing the class with an array.

in the graph class for example, it could be implemented with an adjacency list, or completely with pointers if it was defined exclusively with pointers, an integer index would be meaningless.

anything that doesn't pertain to the class itself.

i.e. a function that would print how vertices have 1 edge, 2 edges, 3 edges, ...

This has nothing to do with the graph class itself, just how ladder.cxx is using it. Thus it should be defined in ladder.cxx, not the graph class.

Note on binary_search and pretty much any other search that is more sophisticated than a simple linear search:

you can ONLY do a binary search on a vector or array that you know is sorted. The reason for this is that the algorithms use comparisons between elements with the assumption that if the target is less than what is being looked at, then the target must occur before the current position.

In the sample input, everything was sorted, but you are not guaranteed for that to happen, unless it is either specified, or you insert everything into your vector in sorted order, thus guaranteeing the fact that your vector is sorted.

Problem 3:

The problem was asking for the subsets of size k, not permutations. A subset can also be seen as a combination. Thus [1,2,3,4] is the same as [1,2,4,3] when looking at subsets. So a program that outputed that was duplicating itself.

Many people wrote a program that found all the permutations, and just didn't output any permutation that wasn't in sorted order.

This does technically work, but will find 1680 permutations to print out 70 subsets, even with the relatively small input of n=8, k=4.

This number will grow very fast, so this type of algorithm is not an efficient one.

See the solution for an example of an algorithm that only computes what it needs to.