Homework 12 (Chapter 13)
The InClass is to be demonstrated (or mailed) to your group leader. To make best use of the time you have in labs, be sure you have
(1) read the assignment before lab
(2) read appropriate background material before lab

The Homework is to be submitted via Eagle. It is to be individual work. You may talk to other students in the course about your design and to get ideas, but you are to write the complete Alice programs by yourself. You may receive help from the Professor, CS Tutors, UTF’s or TA. In your comments, identify (by name) all those who helped you. Failure to do so is considered cheating.

In most cases, it will not be possible to finish the homework during labs. Plan on spending several hours per week outside of class to get the homework done.

Inclass (5 points) (Show to group leader during lab Wednesday or Friday.)

The magician (Local Gallery/people) has learned several tricks. Before he/she begins any magic trick, he/she goes though several moves (a pre-routine of your choosing). One trick he/she knows is how to make objects disappear. However, he/she is absent minded and sometimes tries to make things disappear that have already disappeared. Create a table with multiple objects on it. For the magician object, create a method "makeDisappear" which has an input parameter of "which object" to make disappear. Test the function multiple times to make sure it works properly. When all of the objects on the table have disappeared, make the table disappear.

Write a world function (call it ctShowing) which returns a count of the number of objects which are visible. Print that value using the print command.

Point distribution:
1. (1 point) Magic pre-routine.
2. (2 points) Disappear method
3. (2 points) ctShowing function
4. (1 BONUS point) Write a Boolean function which returns true if the object is taller than it is wide. Test the function.
Hints:

1. Once you start using parameters and variables, the system often refuses to copy the code. ARGGG! This is a bug in Alice that I haven’t found a fix for.

2. Parameters don't have a type associated with them. (This is a bad design.) Thus, you can't tell an object to "walk" (for example) – as not all objects which could be passed in know how to walk. This is a problem. We would like the system to know what kind of operations are legal for the object passed in. Remember that type is both (1) What the object looks like (in terms of storage) and (2) what it can do. Making sure we only use an object for what it is capable of doing is called "type checking". Since the system won't allow you to make a mistake, it will not allow you to call a function/method from a parameter if all possible parameters don't have that function.

The "placeholder technique" can give you a few additional choices. To get the system to take the height of the parameter takes a two step process. Suppose you would like to tell the parameter to “move up” two times “who.height” where “who” is the parameter name. The system doesn’t allow you to do this directly. Instead, you can drag in “anyobject.height” and then afterwards drag “who” to replace “anyobject”.

So for example, in this simple case I want “who” to jump its height. First, I drag in CheshireCat.height (or any object which has a height).

```
world.jump ( [Obj] who)
No variables
who move up ( subject = cheshireCat 's height )
who move down ( subject = cheshireCat 's height )
who turn left 2 revolutions
```

Then I drag the "who" tile to replace “cheshireCat”

```
world.jump ( [Obj] who)
No variables
who move up ( subject = who 's height )
who move down ( subject = who 's height )
who turn left 2 revolutions
```

Since everything has a height, we are confident we have done nothing illegal.

Homework 12 (due at midnight)
More tricks for the magician. Do the following:
1. Write a method "levitate" which takes an object as a parameter. The object moves up its height and then down. If the object is the table, everything on the table should rise (use vehicle). Note, levitate needs to work for objects of any height. You cannot simply "hard code" a specific amount to rise.
2. Write a method "changeVisibility" which makes everything on the table visible again or makes everything invisible depending on the value of the Boolean parameter.

3. Write a method "transform" which takes two objects as parameters. They must be different heights. Call them becomingInvisible and becomingVisible. Move the object becomingVisible to becomingInvisible location, and make becomingVisible invisible (set opacity to 0). Set becomingVisible to the same height as becomingInvisible. (This will require computing the current ratio between the heights.) Slowly change the opacity of the two objects so that becomingInvisible disappears and becomingVisible appears. This gradual disappearing needs to be done in a loop by changing opacity.

4. This magic trick is a variant of the "cutting a person in half" trick. Write a method "cutThrough" which takes a weapon object as a parameter. A person and a knife are originally placed on the table (but are invisible). The method first needs to make all objects invisible (using changeVisibility) and make the person and knife visible. The method drops the knife (Local/kitchen) on the person, the person splits in two pieces and the two pieces leave the stage in different directions. After the person has left the table, the knife and person should disappear again.

Point Distribution
1. (3 points) Levitate
2. (4 points) ChangeVisibility
3. (4 points) Transform
4. (4 points) cutThrough
5. (1 BONUS point) a magic trick of your choice