Organic Molecule Visualization

For many students, organic chemistry is a nightmare that looms in their future, tortures their present, or leaves a dark scar in their academic past. Considered one of the most difficult classes required of science majors or pre-medical students, organic chemistry breaks the confidence of many academically superior students. Why is this subject so dreaded? Unlike biology, it cannot be purely memorized. Unlike inorganic chemistry and physics, its behavior is not dictated by relatively simple mathematical equations. Organic chemistry requires a complete understanding of each topic in order to apply its principles. To make matters worse, the topics of organic chemistry must be learned in a somewhat linear fashion where everything builds on previous knowledge. Due to this aspect, a student failing to grasp a concept one week will have a *much* harder time learning the subject matter of the following week.

Possibly the first task in learning organic chemistry is understanding the names for the scores of molecules which undergo the reactions. The reason a student must *understand* the names (rather than memorize them) is it is practically impossible to recognize every different molecule on sight. Since there are almost limitless molecules with different confirmations, there are certain universal rules for naming them. The IUPAC system of nomenclature dictates what molecules are named. The first task of the Java Organic Construction Set (JOCS) is to take in any alkane name input by the user and display in a two-dimensional format what the structure of the molecule is (i.e. which atoms are bonded together).

The second goal of the JOCS is to help a student visualize what certain molecules truly look like in three-dimensional space. Visualization is a very important aspect of organic chemistry. Unlike general inorganic chemistry, reactions in the organic realm are not constrained purely to the charge and concentration of reagents. Organic molecules can become very large, and their actual shape greatly affects what other molecules they will react with. A model set is almost always required with any organic chemistry course. By combining different atoms with various bonds, a student can see and manipulate a representation of the molecule. Unfortunately, constructing large molecules can be very time consuming. In addition, it is easy to make a mistake in building such structures. The JOCS quickly builds a data structure representing a molecule in memory, then computes the x, y, and z coordinates of each atom as well as the coordinates of each bond that holds two atoms together. The atom can then be displayed on any java-enabled web browser and rotated to view from all angles.

I. Naming and Parsing

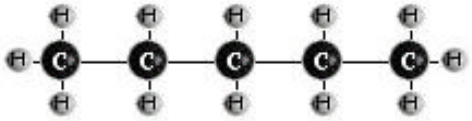
Before learning what is under the hood of the JOCS, it is helpful to know a little

about the IUPAC system of nomenclature and the basic properties of the atoms which compromise organic chemistry molecules. All atoms want to have a full valence electron shell, meaning they are at their most stable when they have no unpaired electrons. Atoms attain this state by sharing their unpaired electrons with atoms that also have unpaired electrons. Two atoms form a bond by sharing one of their electrons with the other. To keep things within the scope of this paper, it is safe to assume carbon forms four bonds, oxygen forms two, and chlorine, fluorine, and hydrogen share one.

Carbon, denoted as C, forms the backbone of every organic molecule. The reason carbon plays such a central role in organic chemistry (and life in general) is its ability to form four bonds relatively easily. With this property, long chains of carbons can form. Each carbon in the chain can form two other bonds, either to different atoms or to other carbon chains. Alkanes, the subject of this program, are simple molecules with no double or triple bonds.

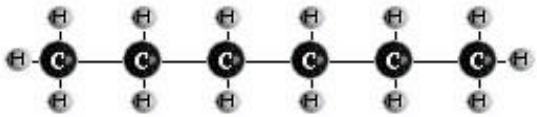
Since carbon is the spine of any organic molecule, the naming system is based on the length of the carbon chain. Greek prefixes are used to represent how long the chain is. These chains are then followed by the suffix *–ane*. The following table shows the first eight prefixes:

Meth	One	Pent	Five
Eth	Two	Hex	Six
Prop	Three	Hept	Seven
But	Four	oct	eight



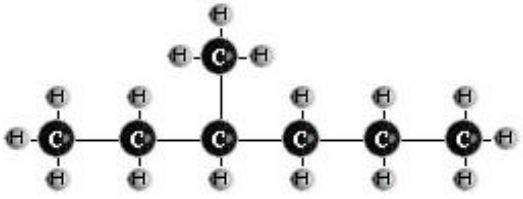
Pentane would then appear as:
Hexane would appear as:

So far, the naming system is simple. However, it becomes more complicated when smaller chains of carbon are attached to the main carbon skeleton. If, for example, there was a chain of length 1 attached to the main chain, it would be denoted as a *methyl*



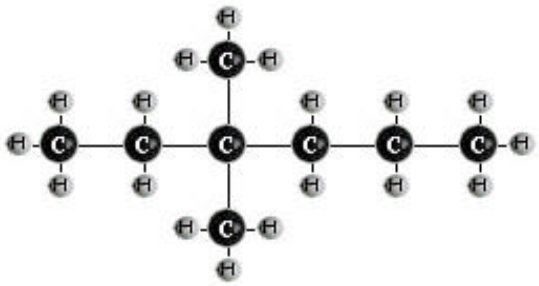
substituent. The number of the main-chain carbon it attaches to denotes the actual placement on the main chain where the substituent is attached. For instance, if there is a methyl substituent attached to the third carbon of the main chain, the resulting molecule is called *3-methyl*hexane. This means there is a methyl (length 1) carbon chain attached to the third carbon of a hexane (length 6) molecule.

3-methylhexane would appear as:



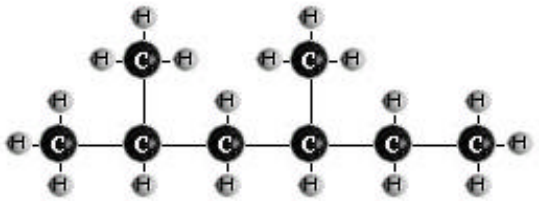
When two methyl groups are attached to the same chain, let us say the third carbon of the main chain in this case, the resulting molecule is called *3,3-dimethyl*hexane.

3,3-dimethylhexane appears as:



Two identical substituents need not be on the same carbon of the main chain.

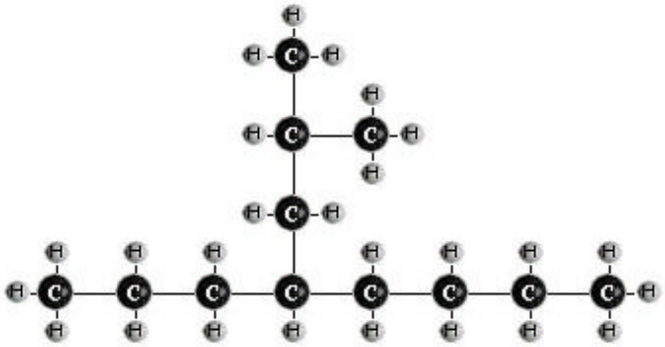
2,4-dimethylhexane appears as:



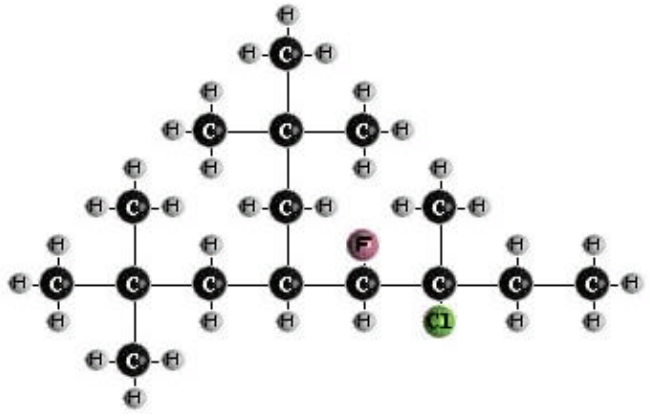
It is important to note that the above molecule is *not* named 3,5-dimethylhexane, the resulting name if the right most carbon is considered carbon 1. This is because the numbering scheme for the main skeleton of alkanes always begins with the end-carbon with the nearest substituent. In this case, the left-most carbon is directly next to the main chain carbon with a substituent, whereas the right-most carbon is further from it's closest substituent.

Chains branching off of the main chain can have their own sub-branches. Sub-branches are represented through the use of parenthesis. For example, an octane chain with a propyl chain attached to it with a methyl group attached to the propyl chain would be named as follows:

4-(2-methylpropyl)-octane:



As can be seen, the molecules can very quickly become complex.
2,2-dimethyl-4-(2,2-dimethylpropyl)-5-flouro-6-methyl-6-chlorooctane:



Parsing names is the first step in constructing a data structure representing the molecule. To simplify things, names are first parsed into a symbolic code easier to build a data structure from. The parsing function, aptly named "parse", is found in the class Mlist. It first searches for any sub-strings containing *di* or *tri* and changes them to an elongated form. For example, 2,2-dimethylpentane is changed into 2-methyl-2-methylpentane. This is done through the following steps:

1) 2,2-**di**methylpentane: *di* is recognized
2) 2,2-di**methyl**pentane: the program searches to the right until the next meaningful prefix is recognized, *methyl* in this case
3) **2,2-di**methylpentane: the sub-string *2,2-di* is replaced by *2-methyl-2-*

The same process is done for *a,b,c-tri prefix* to put it in the form:

a-prefix-b-prefix-c-prefix

Next, the elongated name is scanned from left to right, replacing substituent prefixes with the number represented by the prefix in parenthesis. In addition, any prefix which signifies the length of a chain which has branches on it is put after it's substituents in brackets.

Ex.: 2-methylpentane → 2(1)[5]
4,4-diethyloctane → 4(2)4(2)[8]
4-(2-methylpropyl)octane → 4(2(1)[3])[8]

Any non-carbon-based substituents have special characters to represent them in this code.

Fluorine	f
Chlorine	k
Oxygen	o

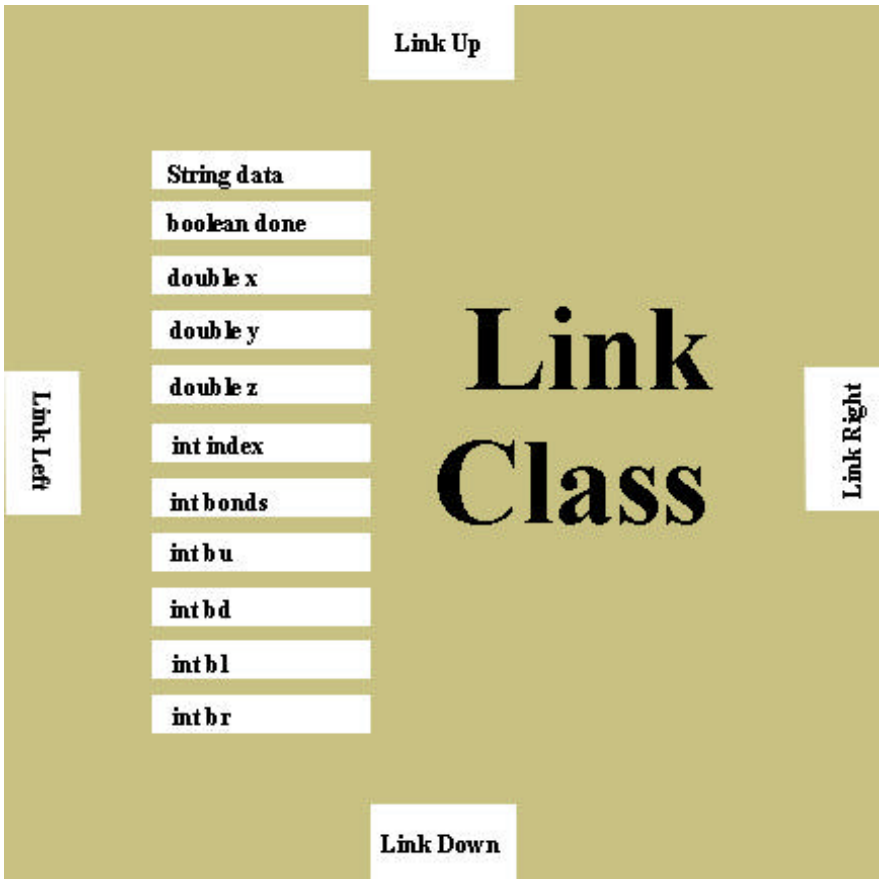
Ex.: 2-chloropentane → 2k[o5]

2,3,4-trifluoroheptane → 2f3f4f[7]

Now that the names are coded, a data structure can be built from them.

II. Building the Data Structure

Each atom in the JOCS molecules is represented by the Link class. This class has many "compartments" for holding all of the data associated with each atom. Below is a diagram of the Link class.



The *data* compartment holds the atom's periodic table representation letters. *X*, *y*, and *z* hold the coordinates of the atom in three-dimensional space. The *index* is merely an integer that is unique for each atom (from 0 to # of atoms). The *bonds* compartment keeps track of the number of atoms the atom is bonded for. *Bu, bd, bl*, and *br* hold the index of atoms the pointers *up*, *down*, *left*, and *right* point to, respectively.

The data structure consists of Links that point to each other (i.e. a doubly linked list). The list is built from the encoded name of the molecule. The structure can be viewed two dimensionally through the use of the MlistTest class, hence the pointers to other links can be thought of as up, down, left, and right in this case.

When the molecule building begins (in the buildMolecule method in the Mlist class), the "direction" the nodes are added is to the right. The directions are represented by the global integers UP, DOWN, LEFT, and RIGHT, which are actually 1, 2, 3, and 4, respectively. When a chain branches off, the direction changes from RIGHT to UP. If

there is already a branch going up and another branch must be added to the same point, the direction changes to DOWN. The following table shows the direction in which nodes are added:

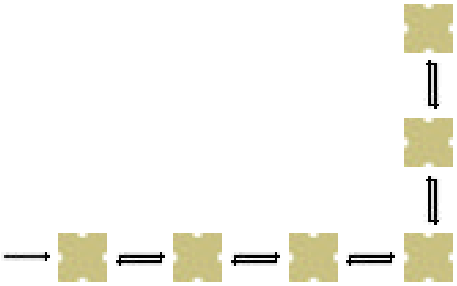
Previous direction	First branch	Second branch
UP	RIGHT	LEFT
DOWN	RIGHT	LEFT
LEFT	UP	DOWN
RIGHT	UP	DOWN

The following is an example of how the carbon skeleton is built.

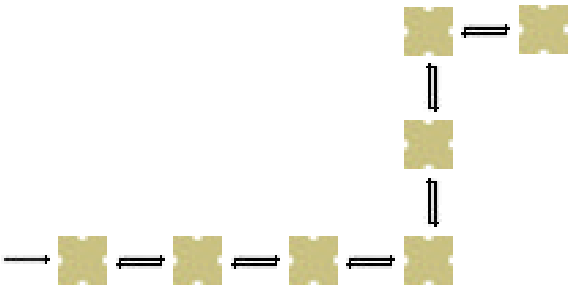
Ex.: Molecule name: 4-(2,2-dimethylpropyl)octane
Coded string representing it: 4(2(1)2(1)[3])[8]

The string is read in character by character. Since the first character is *4*, a chain of length 4 is built to the right:

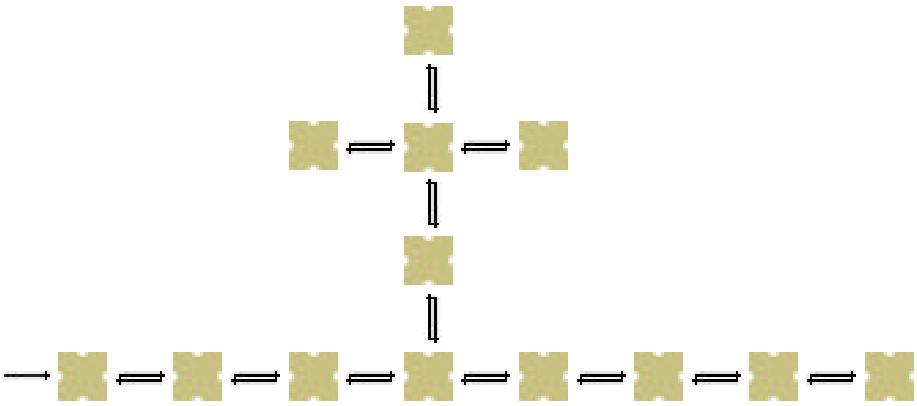
The next character is the open paren *(*. When this is encountered, the buildMolecule method recursively calls itself, passing along the string and the new direction UP. The next character is *2*, so two nodes are added in the upward direction:



After the 2 is read, yet another open paren is encountered. Another recursive call is made, and the direction changes to the right:



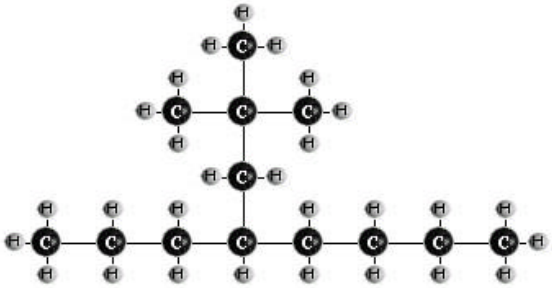
Whenever a close paren is encountered, the point where nodes are being added is returned to the pre-recursive call position. Another branch of length 1 is then added to the left, and the branch of length three, with sub-branches, is complete:



The building process continues to the right until the main chain is length 8, as denoted by *[8]* at the end of the string.

Now that the main skeleton is built, hydrogen must be added to all of the null pointers. This is done in the method addH (in Mlist). Also a recursive function, addH begins at the head, checks to see if it has any null pointers, then links the null pointers to distinct nodes representing hydrogen. Next, addH recursively calls itself to carbon nodes the current one is linked to. Each visited node is marked *true* in the boolean field *done* so infinite recursion does not take place. The *done* field is used frequently in this program, so a separate function, *clearMarks*, changes all of the fields from *true* to *false*.

Once the molecule is built, drawing it in a two-dimensional fashion is a simple matter. The class *MlistTest* has a function *drawMolecule* which imports JPEG images that represent each node in the data structure. The above molecule would appear as:



III. Filling the Coordinates and Bond Information