Project Description

Hydrocarbons are compounds composed of carbon and hydrogen. An **alkane** is a saturated hydrocarbon. Each carbon atom is bound to the maximum number possible: Four atoms. Alkanes can have their atoms arranged in what is called a **normal, straight-chain or unbranched** way.

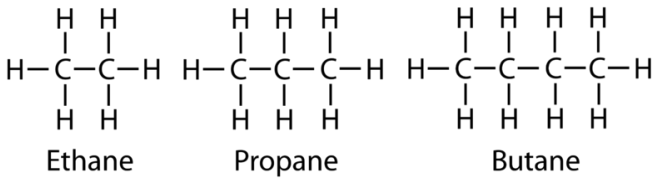

Abbildung 1: Examples of straight-chain alkanes with two, three and four carbon atoms

Alkanes with at least four carbon atoms exhibit **structural isomerism**. This is because they can exist not only as straight-chain molecules, but also in a **branched-chain** structure. For instance, the alkane with four carbon atoms has two isomers and the alkane with five carbon atoms has three isomers.

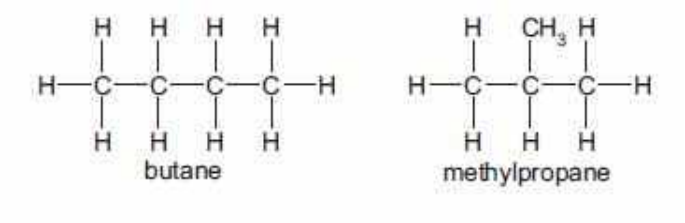

Abbildung 2: C_4H_{10} can exist either in a straight-chain structure (butane) or branched structure with a longest chain of three carbon atoms and a branch of CH_3.

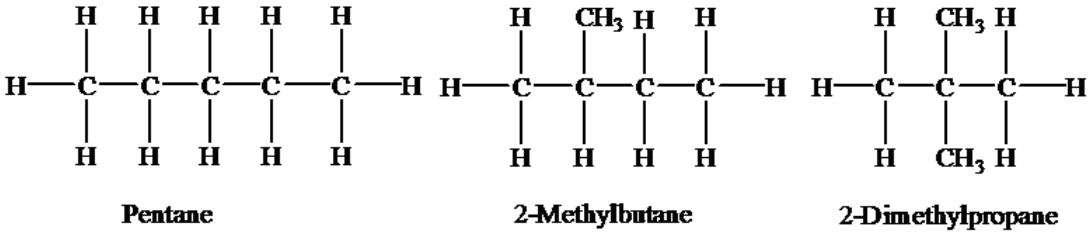

Abbildung 3: The three isomers of C_5H_{12}

While generating the different isomers of some formula, finding the distinct ones can be a bit tricky. Some structures appear to be additional isomers when, in fact, they are identical to already generated ones once rotated and/or mirrored. Your **task** is to develop a system for **alkane isomers generation** according to some specific rules.

Implement the following Prolog predicates according to the given description:

a) The predicate `straight_chain_alkane(N,A)` which succeeds if `N` is a positive integer representing the number of carbon atoms in a straight-chain alkane. `A` is list of length equal to `N`. Each item in that list represents a carbon atom along with its saturated bonds using the following **data structure**:

```
carb(Left,Top,Down,Right)
```

where `Left, Top, Down and Right` are the four atoms that carbon atom is bound to. Note that the two carbon atoms at the end points of a chain are each bound to three hydrogen atoms and one carbon atom.

Important Note: This predicate is not the main requirement of the project. However, it produces important output (the chain list) that serves as a basis for the isomers you will generate in part b).

Test your predicate with the following sample queries:

```
?- straight_chain_alkane(1,A).
A = [carb(h, h, h, h)] ;
false.
?- straight_chain_alkane(2,A).
A = [carb(h, h, h, c), carb(c, h, h, h)] ;
false.
?- straight_chain_alkane(3,A).
A = [carb(h, h, h, c), carb(c, h, h, c), carb(c, h, h, h)] ;
false.
```

b) The predicate `branched_alkane(N,BA)` which succeeds if `N` is a positive integer representing the number of carbon atoms and `BA` is a list representing a branched alkane with `N` carbon atoms. The length of the list `BA` should be `N-1` or less since there is at least one branch attached.

You have to satisfy the following conditions in order to avoid generating isomers that are mere rotations and/or mirrors with respect to the x-axis:

- No branch should be attached to either carbon atom at the end of the chain.
- The number of carbon atoms in any branch should not lead to any new chains that are longer than the current longest chain. However, it is OK if it leads to a chain of a size less than or equal to the current longest chain.
- If a carbon atom has only one branch attached to it, it has to be on top of it. In other words, a carbon atom is allowed to have a down branch attached only if it already has a top branch attached.
- If a carbon atom has two branches attached to it, then the number of carbon atoms in the top branch has to be less than or equal to the number of atoms in the down branch.
- Note that we only care about the number of carbon atoms in a branch not their structure within the branch.

Test your predicate using the following sample queries:

```
?- branched_alkane(3,BA).
false.

?- branched_alkane(4,BA).
BA = [carb(h, h, h, c), carb(c, c1h3, h, c), carb(c, h, h, h)] ;
false.

?- branched_alkane(6,BA).
BA = [carb(h, h, h, c), carb(c, c1h3, h, c), carb(c, h, h, c), carb(c, h, h, c),
carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, h, h, c), carb(c, c1h3, h, c), carb(c, h, h, c),
carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, h, h, c), carb(c, h, h, c), carb(c, c1h3, h, c),
carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, c1h3, c1h3, c), carb(c, h, h, c), carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, c1h3, h, c), carb(c, c1h3, h, c), carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, c1h3, h, c), carb(c, c1h3, h, c), carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, h, h, c), carb(c, c1h3, c1h3, c), carb(c, h, h, h)] ;
false.

?- branched_alkane(7,BA).
BA = [carb(h, h, h, c), carb(c, c1h3, h, c), carb(c, h, h, c), carb(c, h, h, c),
```

```
carb(c, h, h, c), carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, h, h, c), carb(c, c1h3, h, c), carb(c, h, h, c),
carb(c, h, h, c), carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, h, h, c), carb(c, h, h, c), carb(c, c1h3, h, c),
carb(c, h, h, c), carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, h, h, c), carb(c, h, h, c), carb(c, h, h, c),
carb(c, c1h3, h, c), carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, c1h3, c1h3, c), carb(c, h, h, c), carb(c, h, h, c),
carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, c1h3, h, c), carb(c, c1h3, h, c), carb(c, h, h, c),
carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, c1h3, h, c), carb(c, h, h, c), carb(c, c1h3, h, c),
carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, c1h3, h, c), carb(c, c1h3, h, c), carb(c, h, h, c),
carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, h, h, c), carb(c, c1h3, c1h3, c), carb(c, h, h, c),
carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, h, h, c), carb(c, c1h3, h, c), carb(c, c1h3, h, c),
carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, c1h3, h, c), carb(c, h, h, c), carb(c, c1h3, h, c),
carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, h, h, c), carb(c, h, h, c), carb(c, c1h3, c1h3, c),
carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, h, h, c), carb(c, c2h5, h, c), carb(c, h, h, c),
carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, c1h3, c1h3, c), carb(c, c1h3, h, c), carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, c1h3, c1h3, c), carb(c, c1h3, h, c), carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, c1h3, h, c), carb(c, c1h3, c1h3, c), carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, c1h3, c1h3, c), carb(c, c1h3, h, c), carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, c1h3, h, c), carb(c, c1h3, c1h3, c), carb(c, h, h, h)] ;
BA = [carb(h, h, h, c), carb(c, c1h3, h, c), carb(c, c1h3, c1h3, c), carb(c, h, h, h)] ;
false.
```

c) **[Bonus]** The isomers generated by the predicate `branched_alkane` are already filtered to a good extent. However, if you find the project interesting, we encourage you to try to filter your result even further to eliminate isomers that are mirrors with respect to the y-axis as well. The predicate `isomers(N,I)` holds when `N` is a positive integer greater than or equal to 4. `I` is a list of lists containing **only distinct** structural isomers of an alkane with `N` carbon atoms. An isomer where the branching is closer to the start of the list is supposed to be included and its mirror is considered a duplicate and therefore discarded from the answer list.

Test your predicate using the following sample queries:

```
?- isomers(4,A).
A = [[carb(h, h, h, c), carb(c, h, h, c), carb(c, h, h, c), carb(c, h, h, h)],
[carb(h, h, h, c), carb(c, c1h3, h, c), carb(c, h, h, h)]];
false.

?- isomers(5,A).
A = [[carb(h, h, h, c), carb(c, h, h, c), carb(c, h, h, c), carb(c, h, h, c),
carb(c, h, h, h)],
[carb(h, h, h, c), carb(c, c1h3, h, c), carb(c, h, h, c), carb(c, h, h, h)],
[carb(h, h, h, c), carb(c, c1h3, c1h3, c), carb(c, h, h, h)]];
false.

?- isomers(7,A), length(A,Count).
```

```
A = [[carb(h, h, h, c), carb(c, h, h, c), carb(c, h, h, c), carb(c, h, h, c),
carb(c, h, h, c), carb(c, h, h, c), carb(c, h, h, h)],
[carb(h, h, h, c), carb(c, h, h, c), carb(c, c2h5, h, c), carb(c, h, h, c),
carb(c, h, h, h)],
...
Count = 9 ;
false.

?- isomers(10,A), length(A,Count).
A = [[carb(h, h, h, c), carb(c, h, h, c), carb(c, h, h, c), carb(c, h, h, c), ...,
Count = 72 ;
false.
```

Grading and Evaluation

The evaluation of the practical part of the project, i.e the implementation, is the same for the whole team. However, the oral evaluation is team-member-specific. Your oral evaluation will affect your overall project grade. So be prepared :)

Tips

Here is a list of the do's and don't for a better performance in the project as well as in the course:

- Do not think in a procedural way, i.e. do not translate your Java ideas into Prolog code.
- Remember that Prolog is a logic programming language. Sometimes it will be useful to express your rule in natural language first. Maybe then you will see the Prolog code :)
- Work personally on the code. The more you are involved in the coding process, the more you become aware of the advantages and drawbacks of Prolog and the logic programming paradigm as a whole.
- Read carefully the lecture notes and revise the slides. They include a lot of useful information for you, not only concerning technical details but also practical and technical issues.
- Start working early on the project. Do not get carried away and keep track of time. It would be handy if you planed ahead your project and worked out a reasonable timeline.
- Divide the work among your team members. Coordinate your efforts together and meet regularly to update each other with the progress.

Finally, we wish you the best of luck!