2.1 Data objects

33 simple objects

variables

of Prolog

2.1.1 atoms numbers

In chapter 1 we have seen some examples of atoms and variables. In general, however, atoms can take more complicated forms—that is, strings of the following characters:

- upper-case letters A, B, ..., Z
- lower-case letters a, b, ..., z
- digits 0, 1, 2, ..., 9
- special characters such as *, -, <, :, &

Atoms can be constructed in three ways:

1. Strings of letters, digits and the underscore character, _, starting with a lower-case letter:
   - anna
   - nil
   - x25
   - x_25
   - x_2548
   - x_v
   - alpha_beta
   - procedure
   - missJones

2. Strings of special characters:
   - When using atoms of this form, some care is necessary because some strings of special characters already have a predefined meaning. An example is ,:-, .

The topics included are:

- syntax and semantics of basic concepts
- The topics will be covered in the next chapter.

Chapter 1. Here the treatment will be given by its syntactic form. This is different for each type of information (such as data-type declaration or order to recognize the type of an
Atoms, numbers, variables and structures are all terms.
The type of a term can be tested by the following predicates:

- `var(X)` X is a (non-instantiated) variable
- `nonvar(X)` X is not a variable
- `atom(X)` X is an atom
- `integer(X)` X is an integer
- `float(X)` X is a real number
- `atomic(X)` X is either an atom or a number
- `compound(X)` X is a structure
• Terms can be constructed or decomposed:

\[
\text{Term} = \ldots [\text{Functor} \mid \text{ArgumentList}]
\]

functor( Term, Functor, Arity)
arg( N, Term, Argument)
name( Atom, CharacterCodes)
- Terms can be compared:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>X = Y</code></td>
<td><code>X</code> and <code>Y</code> match</td>
</tr>
<tr>
<td><code>X == Y</code></td>
<td><code>X</code> and <code>Y</code> are identical</td>
</tr>
<tr>
<td><code>X != Y</code></td>
<td><code>X</code> and <code>Y</code> are not identical</td>
</tr>
<tr>
<td><code>X =:= Y</code></td>
<td><code>X</code> and <code>Y</code> are arithmetically equal</td>
</tr>
<tr>
<td><code>X =\= Y</code></td>
<td><code>X</code> and <code>Y</code> are not arithmetically equal</td>
</tr>
<tr>
<td><code>X &lt; Y</code></td>
<td>Arithmetic value of <code>X</code> is less than <code>Y</code> (related: <code>=\=</code>, <code>&gt;</code>, <code>&gt;=</code>)</td>
</tr>
<tr>
<td><code>X @&lt; Y</code></td>
<td>Term <code>X</code> precedes term <code>Y</code> (related: <code>@=\=</code>, <code>@&gt;</code>, <code>@&gt;=</code>)</td>
</tr>
</tbody>
</table>
A Prolog program can be viewed as a relational database that can be updated by the following procedures:

```
assert( Clause)  add Clause to the program
asserta( Clause) add at the beginning
assertz( Clause) add at the end
retract( Clause) remove a clause that matches Clause
```

If you want to have a clause in your database and you want to be able to add or remove it dynamically (that is, when you query), you must declare it to be `dynamic` in gprolog with the `:-` `dynamic` designation. The compiler needs this designation when it consults your database.

You can add or remove any clause not already in your database without the `:-` designation.

gprolog does not have `assert/1`. 
• All the objects that satisfy a given condition can be collected into a list by the predicates:

\[
\begin{align*}
\text{bagof}(X, P, L) & \quad \text{L is the list of all } X \text{ that satisfy condition } P \\
\text{setof}(X, P, L) & \quad \text{L is the sorted list of all } X \text{ that satisfy condition } P
\end{align*}
\]

Recall the mathematical definition of a bag compared to the definition of a set. A bag can have duplicates. A set cannot. For example, \(\{a, b, b\} = \{a, b\}\).
- Built-in procedures for reading and writing characters and terms are:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>read(Term)</td>
<td>input next term</td>
</tr>
<tr>
<td>write(Term)</td>
<td>output Term</td>
</tr>
<tr>
<td>put(CharCode)</td>
<td>output character with the given ASCII code</td>
</tr>
<tr>
<td>get0(CharCode)</td>
<td>input next character</td>
</tr>
<tr>
<td>get(CharCode)</td>
<td>input next ‘printable’ character</td>
</tr>
</tbody>
</table>
Summary of $\text{read}(\ X)$

- The next term $T$ is read and matched with $X$.
- If $X$ is a variable, then $X$ is instantiated to $T$.
- If there is no match, the goal $\text{read}(\ X)$ fails with no backtracking.
- If <control-d> is read from the keyboard, or the end of file is reached in a file, $X$ is instantiated to $\text{end_of_file}$.
Figure 6.5 Communication between a Prolog program and several files.
Switching between streams is done by:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>see( File)</td>
<td>File becomes the current input stream</td>
</tr>
<tr>
<td>tell( File)</td>
<td>File becomes the current output stream</td>
</tr>
<tr>
<td>seen</td>
<td>close the current input stream</td>
</tr>
<tr>
<td>told</td>
<td>close the current output stream</td>
</tr>
</tbody>
</table>