Chapter 5
Input and Output

Thus far, the only means we have seen of providing information to a Prolog program has been by asking questions of the Prolog system. Also, the only method of finding out what a variable stands for at some point in the satisfaction of a goal has been by asking a question in such a way that Prolog will print out the answer in the "X = answer" form. Much of the time, such direct interaction with questions is all that is required to ensure that a program is working properly. However, for many occasions it is useful to write a Prolog program that initiates a conversation with you by itself.

Suppose you have a database of world events in the 16th Century, arranged as facts containing dates and headlines. To start with, dates can be represented as integers, and headlines can be represented as lists of atoms. We shall have to enclose some of the atoms in single quotes because they begin with an upper-case letter, and we should not want them to be interpreted as variables:

```prolog
event(1505, ['Euclid', translated into, 'Latin']).
event(1510, ['Reuchlin-Pfefferkorn', controversy]).
event(1523, ['Christian', 'II', flees from, 'Denmark']).
```

Now if we wish to know about a particular date, we could ask a question as follows:

```prolog
?- event(1505, X).
```

and Prolog would print the reply

```
X = ['Euclid', translated into, 'Latin'].
```

Representing the history headlines as lists of atoms confers the advantage that "searches" can be made to find out the date when certain key events happened. For example, consider the predicate we shall define, called `when`. The goal `when(X, Y)` succeeds if `X` is mentioned in year `Y` according to our history headlines:

```prolog
when(X, Y) :- event(Y, Z), member(X, Z).
?- when('Denmark', D).
D = 1523
```
One disadvantage of using lists of atoms is that they are awkward to type in, especially if atoms begin with an upper-case letter. Another possibility, which has advantages and disadvantages of its own, is to represent history headlines as lists of characters. From a previous chapter we know that lists of characters can be typed by putting them between double quotes:

\[\text{event}(1511, "Luther visits Rome").}\]
\[\text{event}(1521, "Henry VIII made Defender of the Faith").}\]
\[\text{event}(1524, "Vasco da Gama dies").}\]
\[\text{event}(1529, "Berquin burnt at Paris").}\]
\[\text{event}(1540, "Reopening of war with Turks").}\]

This is easier to type in, but consider asking the question

\[?- \text{event}(1524, X).\]

If you try this, Prolog will print out an incomprehensible list of ASCII codes corresponding to the characters in the string to which \(X\) is instantiated! Although the list of characters is easy to type in, the question-and-answer sequence is not sufficient to print out a legible answer.

It would be more convenient, instead of asking Prolog questions of this form, to write a program that first asks what date you want to know about, and then prints out the appropriate headline on the computer terminal's display. Headlines could then be represented in any way desired. In order to do these kinds of tasks, Prolog makes available some built-in predicates that print out their arguments on your computer terminal's display. There are also predicates that wait for you to type in text from the computer terminal's keyboard, and instantiate a variable to whatever you typed in. In this way, your program can interact with you, accepting \textit{input} from you, and printing \textit{output} to you. When a program waits for you to type some input from you, we say that it is \textit{reading} the input. Likewise, when a program is printing some output to you, we say it is \textit{writing} the output.

In this chapter we describe various methods for reading and writing. One of our examples will be printing headlines from the history database, and we conclude with a program that accepts normal sentences and converts them into a list of constants that can be processed by other programs. This conversion program, called \texttt{read_in}, can be used as a building block with which to create programs that analyse English language. Such analysis programs are discussed in later chapters, especially Chapter 9.