Figure 4.2
Folding a bird from a square piece of paper

representation is not precisely defined or when information is not given in adequate detail.

Computer science approaches these problems by establishing a well-defined set of building blocks from which algorithm representations can be constructed. Such a building block is called a primitive. Assigning precise definitions to these primitives removes many problems of ambiguity, and requiring algorithms to be described in terms of these primitives establishes a uniform level of detail. A collection of primitives along with a collection of rules stating how the primitives can be combined to represent more complex ideas constitutes a programming language.

Each primitive consists of two parts: its syntax and its semantics. Syntax refers to the primitive’s symbolic representation, and semantics refers to the meaning of the primitive. The syntax of air consists of three symbols, whereas the semantics is a gaseous substance that surrounds the world. As an example, Figure 4.3 presents some of the primitives used in origami.

Naming Items in Programs

In a natural language, items often have multword names such as “cost of producing a widget” or “estimated arrival time.” However, when expressing algorithms in a formal program or in a pseudocode version of a program, multword names can complicate an algorithm’s description. Experience has shown that it is better to have each item identified by a single block of text. Over the years many techniques have been used to compress multiple words into a single lexical unit to obtain descriptive names for items in programs. One is to use underscores to connect words, producing names such as estimated_arrival_time. Another is to use uppercase letters to help a reader comprehend a compressed multword name. For example, one could start each word with an uppercase letter to obtain names such as EstimatedArrivalTime. This technique is often called Pascal casing, because it was popularized by users of the Pascal programming language. A variation of Pascal casing is called camel casing, which is identical to Pascal casing except that the first letter remains in lowercase as in estimatedArrivalTime. This text leans toward Pascal casing, but the choice is largely a matter of taste.
To obtain a collection of primitives to use in representing algorithms for computer execution, we could turn to the individual instructions that the machine is designed to execute. If an algorithm is expressed at this level of detail, we will certainly have a program suitable for machine execution. However, expressing algorithms at this level is tedious, and so one normally uses a collection of "higher-level" primitives, each being an abstract tool constructed from the lower-level primitives provided in the machine's language. The result is a formal programming language in which algorithms can be expressed in a conceptually higher form than in the actual machine language. We will discuss such programming languages in the next chapter.

**Pseudocode**

For now, we forgo the introduction of a formal programming language in favor of a less formal, more intuitive notational system known as pseudocode. In general, a **pseudocode** is a notational system in which ideas can be expressed informally during the algorithm development process.

One way to obtain a pseudocode is simply to loosen the rules of the formal language in which the final version of the algorithm is to be expressed. This approach is commonly used when the target programming language is known in advance. There the pseudocode used during the early stages of program development consists of syntax-semantic structures similar to, but less formal than, those used in the target programming language.

Our goal, however, is to consider the issues of algorithm development and representation without confining our discussion to a particular programming language. Thus our approach to pseudocode is to develop a consistent, concise notation for