We also need to decide how to process the character scanned by the forward pointer; does it mark the end of a token, does it represent progress in finding a particular keyword, or what? One way to structure these tests is to use a case statement, if the implementation language has one. The test

if *forward*↑ = **eof**

can then be implemented as one of the different cases.

3.3 SPECIFICATION OF TOKENS

Regular expressions are an important notation for specifying patterns. Each pattern matches a set of strings, so regular expressions will serve as names for sets of strings. Section 3.5 extends this notation into a pattern-directed language for lexical analysis.

Strings and Languages

The term *alphabet* or *character class* denotes any finite set of symbols. Typical examples of symbols are letters and characters. The set {0,1} is the *binary alphabet*. ASCII and EBCDIC are two examples of computer alphabets.

A *string* over some alphabet is a finite sequence of symbols drawn from that alphabet. In language theory, the terms *sentence* and *word* are often used as synonyms for the term "string." The length of a string s, usually written $|s|$, is the number of occurrences of symbols in s. For example, banana is a string of length six. The *empty* string, denoted ϵ, is a special string of length zero. Some common terms associated with parts of a string are summarized in Fig. 3.7.

The term *language* denotes any set of strings over some fixed alphabet. This definition is very broad. Abstract languages like $\emptyset$, the *empty* set, or $\{\epsilon\}$, the set containing only the empty string, are languages under this definition. So too are the set of all syntactically well-formed Pascal programs and the set of all grammatically correct English sentences, although the latter two sets are much more difficult to specify. Also note that this definition does not ascribe any meaning to the strings in a language. Methods for ascribing meanings to strings are discussed in Chapter 5.

If x and y are strings, then the *concatenation* of x and y, written xy, is the string formed by appending y to x. For example, if x = dog and y = house, then xy = doghouse. The empty string is the identity element under concatenation. That is, $s\epsilon = \epsilon s = s$.

If we think of concatenation as a "product", we can define string "exponentiation" as follows. Define s^0 to be ϵ, and for $i>0$ define s^i to be $s^{i-1}s$. Since ϵs is s itself, $s^1 = s$. Then, $s^2 = ss$, $s^3 = sss$, and so on.

TERM	DEFINITION
prefix of *s*	A string obtained by removing zero or more trailing symbols of string *s*; e.g., **ban** is a prefix of **banana**.
suffix of *s*	A string formed by deleting zero or more of the leading symbols of *s*; e.g., **nana** is a suffix of **banana**.
substring of *s*	A string obtained by deleting a prefix and a suffix from *s*; e.g., **nan** is a substring of **banana**. Every prefix and every suffix of *s* is a substring of *s*, but not every substring of *s* is a prefix or a suffix of *s*. For every string *s*, both *s* and ϵ are prefixes, suffixes, and substrings of *s*.
proper prefix, suffix, or substring of *s*	Any nonempty string *x* that is, respectively, a prefix, suffix, or substring of *s* such that $s \neq x$.
subsequence of *s*	Any string formed by deleting zero or more not necessarily contiguous symbols from *s*; e.g., **baaa** is a subsequence of **banana**.

Fig. 3.7. Terms for parts of a string.

Operations on Languages

There are several important operations that can be applied to languages. For lexical analysis, we are interested primarily in union, concatenation, and closure, which are defined in Fig. 3.8. We can also generalize the "exponentiation" operator to languages by defining L^0 to be $\{\epsilon\}$, and L^i to be $L^{i-1}L$. Thus, L^i is L concatenated with itself $i-1$ times.

Example 3.2. Let L be the set {A, B, . . . , Z, a, b, . . . , z} and D the set {0, 1, . . . , 9}. We can think of L and D in two ways. We can think of L as the alphabet consisting of the set of upper and lower case letters, and D as the alphabet consisting of the set of the ten decimal digits. Alternatively, since a symbol can be regarded as a string of length one, the sets L and D are each finite languages. Here are some examples of new languages created from L and D by applying the operators defined in Fig. 3.8.

1. $L \cup D$ is the set of letters and digits.
2. LD is the set of strings consisting of a letter followed by a digit.
3. L^4 is the set of all four-letter strings.
4. L^* is the set of all strings of letters, including ϵ, the empty string.
5. $L(L \cup D)^*$ is the set of all strings of letters and digits beginning with a letter.
6. D^+ is the set of all strings of one or more digits. □

OPERATION	DEFINITION
union of L and M written $L \cup M$	$L \cup M = \{ s \mid s \text{ is in } L \text{ or } s \text{ is in } M \}$
concatenation of L and M written LM	$LM = \{ st \mid s \text{ is in } L \text{ and } t \text{ is in } M \}$
Kleene closure of L written L^*	$L^* = \bigcup_{i=0}^{\infty} L^i$ L^* denotes "zero or more concatenations of" L.
positive closure of L written L^+	$L^+ = \bigcup_{i=1}^{\infty} L^i$ L^+ denotes "one or more concatenations of" L.

Fig. 3.8. Definitions of operations on languages.

Regular Expressions

In Pascal, an identifier is a letter followed by zero or more letters or digits; that is, an identifier is a member of the set defined in part (5) of Example 3.2. In this section, we present a notation, called regular expressions, that allows us to define precisely sets such as this. With this notation, we might define Pascal identifiers as

letter (**letter** | **digit**) *

The vertical bar here means "or," the parentheses are used to group subexpressions, the star means "zero or more instances of" the parenthesized expression, and the juxtaposition of **letter** with the remainder of the expression means concatenation.

A regular expression is built up out of simpler regular expressions using a set of defining rules. Each regular expression r denotes a language $L(r)$. The defining rules specify how $L(r)$ is formed by combining in various ways the languages denoted by the subexpressions of r.

Here are the rules that define the *regular expressions over alphabet* Σ. Associated with each rule is a specification of the language denoted by the regular expression being defined.

1. ϵ is a regular expression that denotes $\{\epsilon\}$, that is, the set containing the empty string.

2. If a is a symbol in Σ, then a is a regular expression that denotes $\{a\}$, i.e., the set containing the string a. Although we use the same notation for all three, technically, the regular expression a is different from the string a or the symbol a. It will be clear from the context whether we are talking about a as a regular expression, string, or symbol.

3. Suppose r and s are regular expressions denoting the languages $L(r)$ and $L(s)$. Then,

   a) $(r)|(s)$ is a regular expression denoting $L(r) \cup L(s)$.
   b) $(r)(s)$ is a regular expression denoting $L(r)L(s)$.
   c) $(r)^*$ is a regular expression denoting $(L(r))^*$.
   d) (r) is a regular expression denoting $L(r)$.[2]

A language denoted by a regular expression is said to be a *regular set*.

The specification of a regular expression is an example of a recursive definition. Rules (1) and (2) form the basis of the definition; we use the term *basic symbol* to refer to ϵ or a symbol in Σ appearing in a regular expression. Rule (3) provides the inductive step.

Unnecessary parentheses can be avoided in regular expressions if we adopt the conventions that:

1. the unary operator * has the highest precedence and is left associative,
2. concatenation has the second highest precedence and is left associative,
3. | has the lowest precedence and is left associative.

Under these conventions, $(a)|((b)^*(c))$ is equivalent to $a|b^*c$. Both expressions denote the set of strings that are either a single a or zero or more b's followed by one c.

Example 3.3. Let $\Sigma = \{a, b\}$.

1. The regular expression $a|b$ denotes the set $\{a, b\}$.

2. The regular expression $(a|b)(a|b)$ denotes $\{aa, ab, ba, bb\}$, the set of all strings of a's and b's of length two. Another regular expression for this same set is $aa \mid ab \mid ba \mid bb$.

3. The regular expression a^* denotes the set of all strings of zero or more a's, i.e., $\{\epsilon, a, aa, aaa, \cdots\}$.

4. The regular expression $(a|b)^*$ denotes the set of all strings containing zero or more instances of an a or b, that is, the set of all strings of a's and b's. Another regular expression for this set is $(a^*b^*)^*$.

5. The regular expression $a \mid a^*b$ denotes the set containing the string a and all strings consisting of zero or more a's followed by a b. □

If two regular expressions r and s denote the same language, we say r and s are *equivalent* and write $r = s$. For example, $(a|b) = (b|a)$.

There are a number of algebraic laws obeyed by regular expressions and these can be used to manipulate regular expressions into equivalent forms. Figure 3.9 shows some algebraic laws that hold for regular expressions r, s, and t.

[2] This rule says that extra pairs of parentheses may be placed around regular expressions if we desire.