String patterns in SNOBOL4 are expressions that can be assigned to variables. For example, consider the following:

```
LETTER = 'abcdefghijklmnopqrstuvwxyz'
WORDPAT = BREAK(LETTER) SPAN(LETTER) . WORD
```

`LETTER` is a variable with the value of a string of all lowercase letters. `WORDPAT` is a pattern that describes words as follows: First skip until a letter is found, then span those letters until a nonletter is found. This pattern also includes a "." operator, which specifies that the string that matches the pattern is to be assigned to the variable `WORD`.

This pattern can be used in the statement

```
TEXT WORDPAT
```

which attempts to find a string of letters in the string value of the variable, `TEXT`.

Another language that includes built-in pattern matching operations is Perl. In this case, the pattern-matching expressions are somewhat loosely based on mathematical regular expressions. In fact, they are often called regular expressions. They evolved from the early UNIX line editor, **ed**, to become part of the UNIX shell languages. Eventually, they grew to their most complex form in Perl. It takes an entire chapter of a Perl book to explain these expressions. In fact, there is now a complete book on this kind of pattern-matching expressions (Friedl, 1997). In this section, we provide only a brief look at the style of these expressions through two relatively simple examples. Consider the following pattern expression:

```
/[A-Za-z][A-Za-z\d]+/
```

This pattern matches (or describes) the typical name form in programming languages. The brackets enclose character classes. The first class specifies all letters; the second specifies all letters and digits (a digit is specified with the abbreviation `\d`). If only the second class were included, we could not prevent a name from beginning with a digit. The plus operator following the second category specifies that there must be one or more of what is in the category. So, the whole pattern matches strings that begin with a letter, followed by one or more letters or digits.

Next consider the following pattern expression:

```
/\d+\.?\d*|\.\d+/
```

This pattern matches numeric literals. The `\.` specifies a literal decimal point. The question mark quantifies what it follows to have zero or one appearance. The vertical bar (`|`) separates two alternatives in the whole pattern. The first alternative matches strings of one or more digits, possibly followed by a decimal point, followed by zero or more digits; the second alternative matches strings that begin with a decimal point, followed by one or more digits.

Perl-style pattern matching was recently added to JavaScript.