Table 2.4. Algorithm Used by the LEX State-Machine Driver

```
current_state = 0;
previously_seen_accepting_state = none_seen;

if( lookahead character is end-of-input )
    return 0;

while( lookahead character is not end-of-input )
{
    if( there is a transition from the current state on the current lookahead character)
    {
        current_state = that state;
        advance the input;

        if( the current state is an accepting state )
        {
            remember the current position in the input
            and the action associated with the current state;
        }
    }
    else
    {
        if( no accepting state has been seen )
        {
            There's an error:
                Discard the current lexeme and input character.
                Current_state = 0;
        }
        else
        {
            back up the input to the position it was in when it saw the last accepting state
            perform the action associated with that accepting state;
        }
    }
}
```

Disadvantages of greedy algorithm.

Note that the greedy algorithm does have its disadvantages: It's tricky to implement and tends to be relatively slow. It can also cause the recognizer to behave in sometimes unexpected ways. (The LEX input expression `(\n|.)*` tries to absorb the entire input file, for example.) It is nonetheless the best (and sometimes the only) choice in most real lexical-analysis applications.

The nongreedy algorithm (matches first string).

The second type of algorithm (the *nongreedy* algorithm) is much simpler. Here, the shortest possible input string is recognized, and the machine just accepts as soon as an accepting state is entered. A nongreedy recognizer program is much simpler to implement than a greedy one, and is much faster as well. Nonetheless, this algorithm can be used only when all the accepting states in the machine are terminal nodes—when they have no outgoing transitions.

Terminal nodes.