Problem 29: Finite State Automata[1]

Source filename:	`fsa.(cpp|java)`
Input filename:	`fsa.in`
Output filename:	`fsa.out`

A finite state automaton (fsa) is a common computation structure used in recognition of strings of symbols. For each fsa, the set of strings it recognizes is called the language accepted by the fsa. The purpose of the fsa is to read an input string and output either `accept` (in which case the string is in the language recognized by this fsa) or `reject` (in which case the string is not in language recognized by the fsa).

The set of valid input symbols for a fsa is called the **input alphabet**. Each fsa has its own input alphabet. The fsa consists of a set of states and a description of how the fsa changes from one state to another. The structure of a fsa can be drawn as a **transition diagram**.

A transition diagram is a picture of the states of the fsa with labeled arcs connecting the states. The arcs are directed and define how the fsa changes from one state to another. The states of a fsa are of three types. There is one **start state**. This is the state the fsa begins in when starting to read an input string. There is a set of states defined as **accept states**. This set may or may not include the start state. The third set of states can be classified as **reject states**.

A fsa operates by beginning in the start state and reading the input string – one symbol at a time. When a symbol is read, it is compared to the labels on the arcs leaving from the current state. The fsa changes to the new state pointed by the arc whose label matches the current input symbol. This process continues until 1) the end of the input string is encountered or 2) an input symbol is encountered for which there is no matching label on any output arc for the current state. In the first case, if the last sate of the fsa is an accept state, the fsa is said to accept, or **recognize**, the input string and should output: `accept`. This means that the input string is in the language recognized by the fsa. If the final state is not an accept state or if an input symbol does not match any of the labels on any output arc leaving the current state, the fsa rejects the input string and should output: `reject`.

In figure 1 below illustrates a sample fsa. The input alphabet is the set of numeric digits, {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. The language recognized by this fsa is the set of all strings of digits representing even integes. The start state is indicated by 'S'. In this example, there is only one accept state and it happens to the start state. The accept states are drawn with a thicker line than the reject states. Also notice that arcs may have multiple labels. The symbols are separated by commas. Thus, if the current state is 'S', the machine would change to state '1' if the next input symbols is 1, 3, 5, 7, or 9.

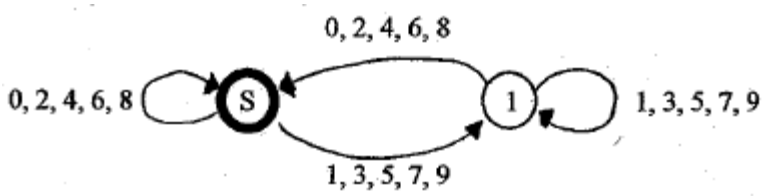

Figure 1

Suppose we read an input string '124' for the previous fsa example. The fsa would start in the start state, S. When the first input symbol is read ('1'), the fsa should follow the bottom arc from state S to state 1. After reading the next input symbol, '2', the fsa should follow the top arc from state 1 back to state S. Finally, the

[1] This problem appeared on an ACM South Central Regional Programming Contest.

last input symbol, '4', would cause the fsa to loop from state S back to state S. The fsa has now encountered the end of the input string, so the fsa outputs `accept` since its final state is an accept state.

If "26762" is read as input to the previous fsa, the sequence of states it would pass through, beginning with the start state is S S S 1 S S. Since the final state is an accept state, the fsa should output `accept`. However, if the input were "1435", the sequence of states is S 1 S 1 1 and, not ending in an accept state, the fsa should output `reject`.

Notice that since in Figure 1 the start state is an accept state, this fsa recognizes zero length strings. If we want to define a new fsa that recognizes even numbers where there must be at least one digit, we can define it as:

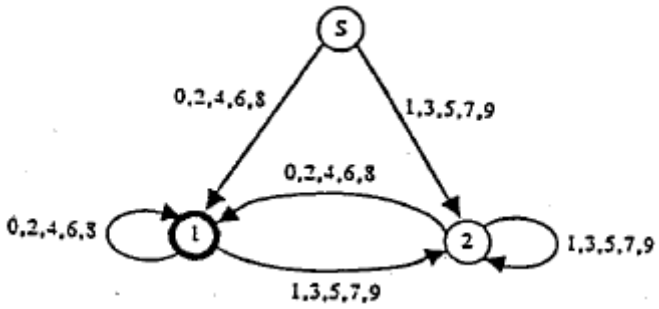

Figure 2

In this example, the accept state is the state '1'. If the input string has no characters at all, the final state is the start state, but since the start state is not an accept state in this example, the fsa would reject a zero length string. Otherwise, this fsa works the same as the one in Figure 1.

The next example recognizes strings beginning with an `a` followed by any number of `b`'s, and ending with a single `c`. The input alphabet is {a, b, c}.

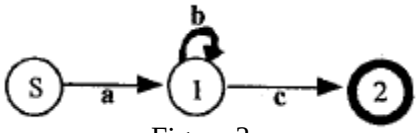

Figure 3

Here, if "`abbbc`" is the input, the sequence of states is S 1 1 1 1 2, and the output is `accept`. If "`ababbc`" is the input string, the sequence of states begins with S 1 1; however, when the fsa encounters the second `a` in the input string, it is in state '1' and there is no arc leaving state '1' labeled with an `a`, so the fsa fails to recognize this string and outputs: `reject`.

Your problem is to write a program that simulates a fsa that recognizes floating point constants (as defined in the Pascal programming language). The input alphabet is {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ., +, -, E, e}. (The character that follows the 9 in the preceding list is a decimal point.) In the transition diagram in Figure 4, the label digit refers to any of the numeric digits 0 through 9.

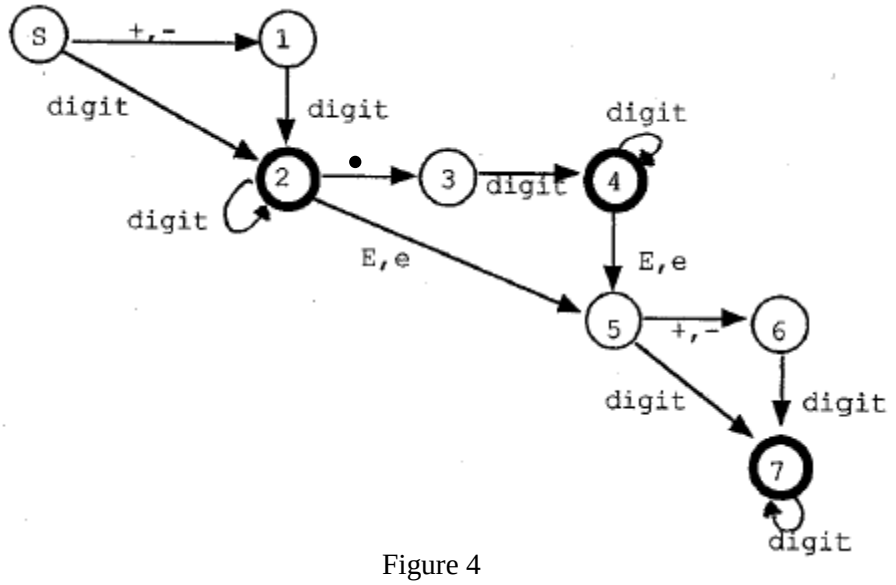

Figure 4

As your program reads an input string from the input file, it should output each new state as it is entered, separating the state numbers by a single space. It should always output the start state S first. Then, after the fsa finishes for a given input string, it should output either accept or reject after skipping 2 spaces.

The end of the input file is marked by a line that contains a single asterisk.

Example Input File (fsa.in)

```
-55.7E1
4
-578
0.25
.25
5434.25e
*
```

Example Output File (fsa.out)

```
S 1 2 2 3 4 5 7  accept
S 2  accept
S 1 2 2 2  accept
S 2 3 4 4  accept
S  reject
S 2 2 2 2 3 4 4 5  reject
```