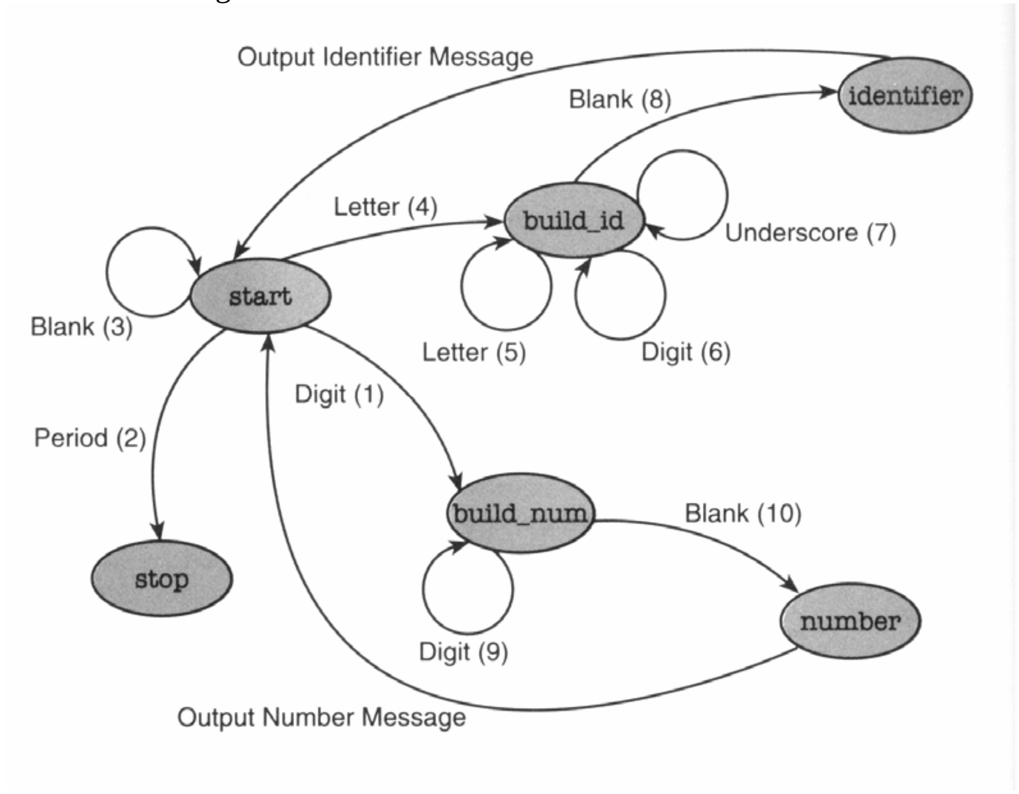
FINITE STATE MACHINES

A finite state machine (FSM) consists of a set of states, a set of transitions, and a string of input data. In the FSM shown below, the named ovals represent states, and the arrows connecting the states represent transitions. The FSM is designed to recognize a list of C++ identifiers and nonnegative integers, assuming that the items are ended by one or more blanks, and that a period marks the end of all the data. The trace below shows how the diagrammed machine would process a string composed of one blank, the digits 9 and 5, two blanks, the letter K, the digit 9, one blank, and a period. The machine begins in the start state.



Here is a trace of the execution of this FSM when given the data: `" 95  K9 ."`

State	**Next Character**	**Transition**
start	‘ ‘	3
start	‘9’	1
buildNum	‘5‘	9
buildNum	‘ ‘	10
number		Output number message
start	‘ ‘	3
start	‘K’	4
buildId	‘9’	6
buildId	‘ ‘	8
identifier		Output identifier message
start	‘.’	2
stop		

Consider writing a program that uses an enumerated type to represent the names of the six states. Your program should process a correctly formatted line of data, identifying each data item. Here is a sample of correct input and output. Of course, your output should be even prettier… And you should consider what happens if your program receives BAD input, like " `3#Xy! 4,5-Z .`"

Input:

```
rate R2D2 48     2 time       555666  .
```

Output:

```
rate - Identifier
R2D2 - Identifier
48 - Number
2 - Number
time - Identifier
555666 - Number
```

Use the following code fragment in your main program (or, even better, in a driver function called `FiniteStateMachine` that is called by `main`), and design function `Transition` to return the next state for all the enumerated transitions of the finite state machine. If you include the header file `ctype.h`, you can use the library function `isdigit` which returns 1 if called with a digit character, 0 otherwise. Similarly, the function `isalpha` checks whether a character is a letter. When your program correctly models the behavior of the FSM shown, extend the FSM and your program to allow optional signs and optional fractional parts (i.e., a decimal point followed by zero or more digits) in numbers.

```
STATE  currentState;              // STATE is an enumeration…
char   cInputChar;                // the next input character read

currentState = STATE_START; // begin in the "start state"

do
{
     // handle the two "null transition" cases: identifier, number

     if (currentState == STATE_IDENTIFIER)       // have identifier
     {
          cout << " - Identifier" << endl;
          currentState = STATE_START;
     }

     else if (currentState == STATE_NUMBER) // have a number
     {
          cout << " - Number" << endl;
          currentState = STATE_START;
     }

     cInputChar = cin.get ();                    // get next char

     if (cInputChar != ' ')                      // echo if not WS
          cout << cInputChar;

     currentState = Transition (currentState, cInputChar);

} while (currentState != STATE_STOP);            // loop 'til done
```