**Programming Paradigms: Overview to State Oriented**

There are many ways to view programming!

Four Principal Programming Paradigms are:

**COP: Control Oriented**
- Emphasizes the flow of actions
- Concentrates on actions
  - imperative
  - procedural

**DOP: Data Oriented**
- Emphasizes the flow of data
- Concentrates on data (& methods)
  - data flows: pass, hide & share
  - data space: store space access

**OOP: Object Oriented**
- Emphasizes both data and actions
- Concentrates on Class & objects
  - integrates actions & data
  - simulates interacting systems

**SOP State Oriented**
- Concentrates on States and transitions between them
**SOP: State-Oriented Programming**

State-oriented programming is a programming paradigm which emphasizes the state (data values) and the transitions (or changes) between states. It is most useful when inputs come in a sequence, such as a string of characters, or a stream of coins.

*State diagrams* (sometimes called finite state machines, sequential machines, or regular automata) are useful to describe such systems. State diagrams consist of circles or rounded boxes representing states and arrows indicating the transitions (changes) between states, as shown in the figure.

Every transition arrow from one state \( S \) to the next state \( S' \), must have an input, and often has a corresponding output. The input and output are separated; sometimes by a slash, or other times the input is above the arrow and the output is below it.

**Dispense15** of the given figure shows the behavior of an algorithm which accepts sequences of nickels (5 cent pieces) and dimes (10 cent pieces), and outputs an item when the accumulated sum reaches 15 cents. The states in this case represent the amounts of money (0, 5, and 10 cents) accumulated at any instant of time. The initial state, shown by a dotted arrow, is state 0. Then when a nickel is input, the state changes to 5 (without putting out an item). When the state is 10, and a nickel is input then an item is output and the next state is 0.

Transitions, or state changes, caused by input sequences of coins can be shown by a trace as given below. Notice that there is no change output by this machine, so that when the state is 10 and a dime is input, then an item is output and the next state is 5 (crediting the extra 5 cents to the next transaction). That is not very friendly! An alternate dispensing machine, which outputs change, is shown next.
Dispense 15 with Change, of the given figure is a modification of the previous dispenser, but this version also outputs change. For example, when in state 10, and a dime is input, an item is output and change (of 5 cents) is output also, and the next state is 0.

Tables of transitions also describe such sequential systems, as shown below to the right. For each of the 3 states there are 2 possible coins input thus yielding 6 possible combinations. Such representations help check that all possibilities are considered.

Programs corresponding to such state diagrams are easy to do, as follows. They consist of a large choice, corresponding to the states, and then for each state is a sub choice depending on the input. Notice that the repeat on false causes the loop to continue forever.

Set state = 0

Repeat
ExitOn (false) -- loop forever!

Input coin -- enter, prompt, echo

If (state == 0) then
  If (coin == 5) then
    Set state = 5
  Else -- coin is 10
    Set state = 10
  EndIf

ElseIf (state == 5) then
  If (coin == 5) etc

Else -- (state is 10)
  Output item, chg, reset, etc
EndIf

EndRepeat
**Count3: Divide by 3**

Count3 is an algorithm which has an input of 0 and 1 only, and outputs a value of 1 only after an input of 3 1s.

The state diagram shows three states having values 0, 1, 2 and 3. The state represents the number of accumulated 1s.

For example, an input/output sequence follows. Only at the third input of a 1, does the output create a 1.

```
0 1 1 1 0 1 0 0 1 0 1 0 1 0 1 1
0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1
```

Count2, Count4, Count5, and many others are similar.

PROBLEM: Create the state diagram for a Count2 and a Count4.

---

**Add2: Sequential Adder**

Add2 sums any two numbers, given in binary. The two numbers are input with least significant digits entered first, as a pair of inputs.

The state $c$ is the value of the carry; the output $s$ is the sum at each stage.

For example the numbers 6 and 12 (which in binary are $0110$ and $1100$) sum to $10010$ which in decimal, base 10, is 18

- $a = 0110 = 6$
- $b = 1100 = 12$
- $s = 10010 = 18$

A trace of the above addition follows; note that the least significant digits are entered first.
Optimizing a state system is simple when viewed as a table with input $x$ and output $y$. The previous Count3 system is shown at the left as a state diagram and a state table. Notice that the rows related to state 0 and 3 are identical, so they may be combined. The diagram and table to the right show the equivalent but simpler state system.

![State Diagram and Table](image)

DeBlank Stream

Suppose it is necessary to remove extra blank characters from a string of characters. For example, the stream

"If it is, so be it!"

is converted to:

"If it is, so be it!"

A state diagram for this follows: the inputs are two kinds of characters, a space $s$ and any other character $n$, which is not a space. The state is either in (within a word) or out (outside of a word), with transitions between the states given in the state diagram. Notice that the output echoes the input except in one particular transition.

![State Diagram](image)
// Does remove extra blanks
String str;
String ch; //single character
boolean within; //state of ch in str
String SPACE;
SPACE = " ";
str = "If it is, so be it!";
System.out.println (str);

within = true;
for (int i=0; i < str.length(); i++) {
    ch = str.substring(i, i+1);
    if (within) {
        if (ch.equals(SPACE)) {
            within = false;
        }
        System.out.print (ch);
    } else { //state is outside
        if (!ch.equals (SPACE)) {
            within = true;
            System.out.print (ch);
        }
    }
}

Problems on SOP: State Oriented Programming

Dispense 20
Draw the state diagram describing the dispenser of 20-cent items (with change)
if inputs are sequences of nickels (5 cents) and dimes (10 cents), with only one
coin entered at a time.
Represent this dispenser in another way also.

Dispense 20 again
Draw the state diagram for dispensing 20-cent items (with proper change)
if inputs are nickels, dimes, and quarters.

Dispense 30
Draw the state diagram for dispensing 30 cent items (with proper change)
if inputs are nickels, dimes and quarters.

Dispense 7
Draw the state diagram for dispensing 7-cent items (with proper change)
if inputs are nickels and pennies.

Parity
Draw a state diagram for a system having inputs of 0 and 1 only,
which indicates whether an odd number of inputs had value 1.

More Counts
Create the state diagram for a Count2 and a Count4.
MORE PROBLEMS ON SOP

**WordCount**
Modify the previous DeBlank code to count the words in a sentence.
Notice that words are separated by blanks.

**UnComment**
Create an algorithm and code which removes comments from text.
Assumed that text has only one kind of comment, /* and */.

**BinarySubtract**
Create an algorithm and code which subtracts two binary numbers,
when entered with least significant digits first.

**Decimal Adder**
Create an algorithm and code which adds two base ten numbers,
when entered with least significant digits first.