This model generates an infinite series of values in the variable x. It executes the commands represented generically by BOX for each value of x. The variable x is the LOOP VARIABLE and often has some other name.

```
INITIALIZE
while (true) {
  BOX
  ADVANCE
}
```

INITIALIZE should be one or more steps giving the STARTING values for the loop variable (or variables).

ADVANCE should use current loop variable values to compute the NEXT loop variable values. ADVANCE can be one step or a complex combination of steps.
Problem: Generate the sequence 5,7,9,11,13,... in the variable named w used by the commands represented by BOX.

Analysis: Sequence is generating by adding 2 to each number to get the next one. First element is 5. The ... at the end means an INFINITE sequence:

Solution:

```
w = 5;
while(true) {

BOX

w = w+2;
}
```

First value for w

Makes sequence INFINITE

Generic - anything could go here

Each element is previous one plus 2

INITIALIZE

ADVANCE
Consider this sequence generated in variable $x$:

$$1,2,3,4,5,10,15,20,25,...$$

If this were JUST an arithmetic sequence it would go either:

$$1,2,3,4,5,6,7,8,9,10,...$$

Or:

$$0,5,10,15,20,25,...$$

But instead the stepping behavior changes part way through the sequence. Namely the stepping is just by 1 for those $x$ values LESS THAN 5 but switches to 5 for the remaining $x$ values.

Hence the ADVANCE part of the loop has to be an if-else statement.

The solution is shown on the next page.
x = 1;

while(true) {
    BOX
    
    if(x<5) x=x+1;
    else x=x+5;

}
Consider this sequence generated in variable x:

\[1,2,4,5,7,8,10,11,13,14,\ldots\]

If this were JUST an arithmetic sequence it would go either:

\[1,2,3,4,5,6,7,8,9,10,\ldots\]

Or:

\[2,4,6,8,10,\ldots\]

But instead the stepping behavior alternates between two different steps sizes. It starts with stepping by 1 and then switches to stepping by 2, and then back to stepping by 1, etc.

Thus the step size is ITSELF a loop variable - name it \(w\). The loop variable \(w\) starts off as 1 - as already explained. To get its value to alternate between 1 and 2, a TRICK is used:

\[w = 3 - w;\]

Which works because 3-1 gives 2 but 3-2 gives 1. The solution is shown on the next page.
x = 1;
w = 1;

while(true) {
    x = x+w;
    w = 3-w;
}

INITIALIZE
Requires TWO variables to be initialized

ADVANCE
Generated variable x is advanced using w.
Step variable w is switched using subtraction trick
Suppose we want the sequence:

\[4, 9, 16, 25, 36, 49, 64, \ldots\]

to be generated in the variable \( x \).

The trick here is to reformulate the sequence so as to expose another simpler sequence HIDING inside the given sequence:

\[2 \times 2\ ,\ 3 \times 3\ ,\ 4 \times 4\ ,\ 5 \times 5\ ,\ 6 \times 6\ ,\ 7 \times 7\ ,\ 8 \times 8\ ,\ \ldots\]

The simpler sequence should be generated in ANOTHER loop variable - let us call it \( w \). We want \( w \) to go:

\[2, 3, 4, 5, 6, \ldots\]

and that is easy.

Note that \( x \) will NOT have an ADVANCE. Each time through the loop, \( x \) is DIRECTLY computed from the "hidden" sequence variable \( w \) and ONLY \( w \) is ADVANCED.

The Code is on the next page.

NOTE: Java does not have a power operator so that

\[\text{power}(w, 2)\]

Is used instead of "\( w \) squared".
w = 2;

while(true) {

    x = power(w,2);  

    BOX

    w = w+1;

}

ONLY w is INITIALIZED

Compute x BEFORE using it in the BOX commands

ONLY w is ADVANCED