EXAMPLE 5 – A simple "While" loop with a Decision (Nested control structures)

Problem: We will write a program to inspect a column of positive integers, where we do not know in advance how many numbers there are, and determine how many are even and how many are odd.

Analysis:

We can build on Example 4, where we already have the machinery for inputting positive integers and detecting when the input is done. Instead of accumulating the sum, we will set up counters to count the even integers and the odd integers. (Recall that an even integer is one that is divisible by two with zero remainder, and an odd integer is an integer that is not even.)

We will set up two variables EvenCount and OddCount, and start them out at zero. For each positive integer encountered in the input, we will add one to one of these two counters.

Variables table:

Name	Data Type	Usage
NumberEntered	Integer	Any of the numbers entered by the user
EvenCount	Integer	Counts the even inputs
OddCount	Integer	Counts the odd inputs

Test Cases:

Numbers entered	EvenCount	OddCount
0	0	0
7, 0	0	1
8, 0	1	0
3,4,5,6,8,0	3	2
8,7,4,2,4,7,1,0	4	3

Pseudocode

```
Begin program
Declare Integer NumberEntered, EvenCount, OddCount
// Setup
Set EvenCount = 0
Set OddCount = 0
Print "Start entering positive integers, then enter zero when you are done.\n"
Input NumberEntered

// Start collecting data and counting evens and odds   [Continued on next page…]
```

```
While (NumberEntered > 0)
        If ( (Remainder of NumberEntered on division by 2) == 0)
                EvenCount = EvenCount + 1
        Else
                OddCount = OddCount + 1
        End If
        Input NumberEntered        // Now we get the next number!
End While

Print "The number of even positive integers entered is  " + EvenCount
Print "The number of odd positive integers entered is  " + OddCount
End Program
```

Note 1

Here we have one control structure nested inside another: an If…Endif inside a While..EndWhile. This is very common. In principle, one can have structures nested to any depth, although experienced programmers avoid nesting to any depth greater than three – it gets too error-prone.

There is a fundamental principle illustrated here, that has very broad applicability: *Everything that has a beginning must have an end.* In pseudocode, we usually use matching pairs of words or phrases. In C and related languages, the braces { … } serve the same purpose, albeit in a very different style.

Note 2

Consider the condition

```
        If ((Remainder of EvenCount on division by 2) == 0) …
```

That's great pseudocode, but what is a programmer doing to do with it.? Some languages have an operator symbol for doing a division, discarding the quotient, and keeping only the remainder. In C (and in Java, Perl, and many other languages), the symbol % is used for this purpose. So in C, the above condition would be written

```
        if (EvenCount % 2 == 0) …
```

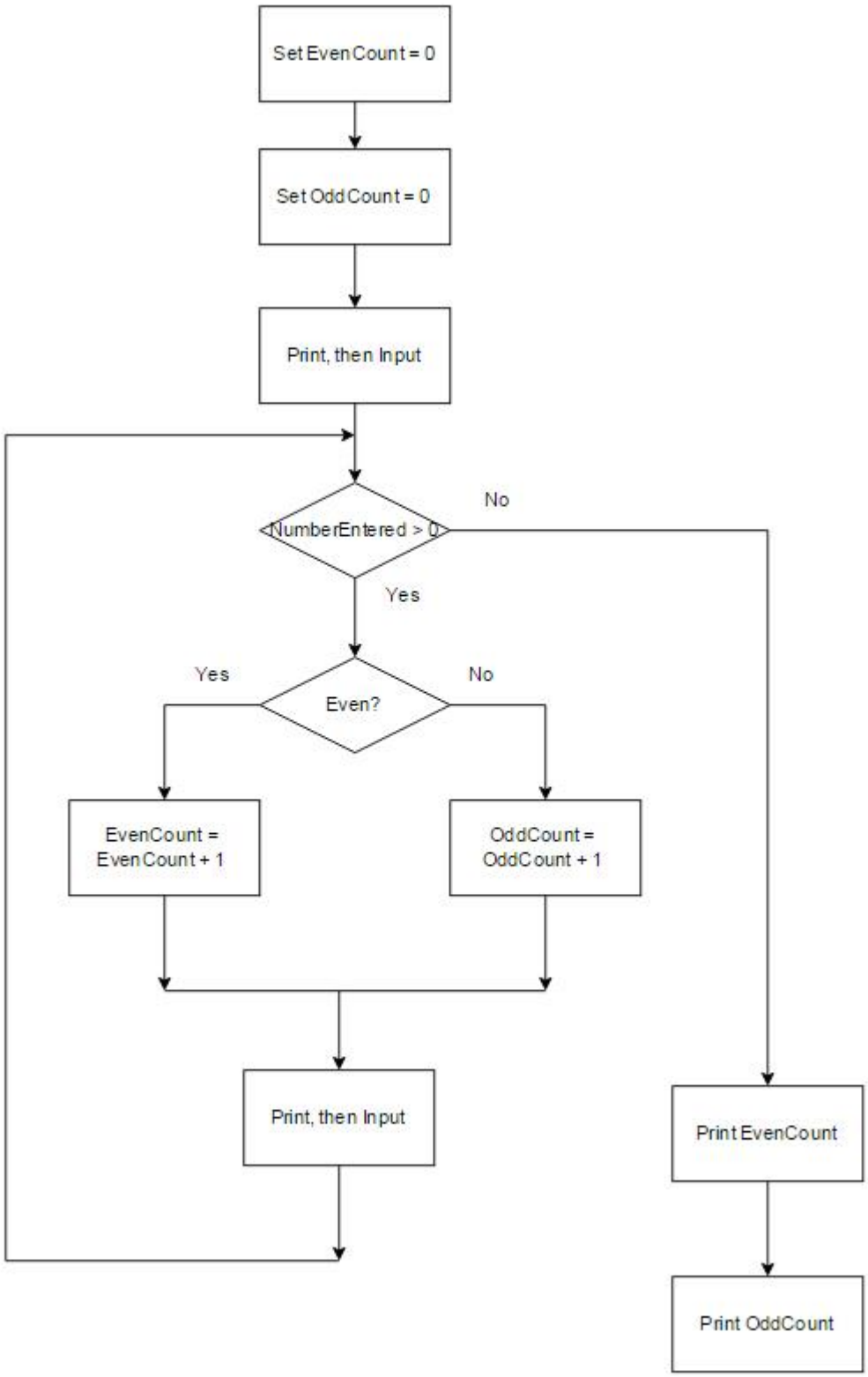
In languages that do not provide a remainder operation, the following trick for determining if an integer is even works nicely:

```
        If (2*(EvenCount/2) == EvenCount) …
```

Why does this trick work?

See the Flowchart on the Next Page

You will notice that I have taken some liberties with what is written in the pseudocode, abbreviating some names and even putting a Print and an Input in the same box. And I have not included declarations. Remember that the purpose of a flowchart is to present a good visual representation of the control flow!


Set EvenCount = 0
Set OddCount = 0
Print, then Input
NumberEntered > 0
No
Yes
Yes
Even?
No
EvenCount = EvenCount + 1
OddCount = OddCount + 1
Print, then Input
Print EvenCount
Print OddCount