FOR – NEXT

This is also a simple loop similar to START-NEXT loop, but now we have a variable that acts as a counter the name of that variable has to be right after the FOR, the syntax is like this:

InitialValue FinalValue

FOR

  CounterVariable

  [body of the loop]

NEXT

When the program reaches FOR, the CounterVariable is initialized to InitialValue Then the body of the loop is executed and when the NEXT is reached CounterVariable increments in 1 and if its value is less or equal than FinalValue the control return to START and the loop is executed again, if the CounterVariable value is greater than FinalValue the control exits the loop and continue to the next statement in the program after NEXT. If the FinalValue is less or equal than the InitialValue the loop is executed just once. CounterVariable is a local variable that can be used only inside the loop.

Example: Given an integer n, this program find the sum of the first n integers, we are not going to use a variable to find this sum, instead, we are going to initialize a number to zero and keep it on the stack.

```
«
 0 1 ROT    @ Initalice the sum to 0, InitialValue to 1 and FinalValue to n
 FOR        @ Starts the loop
  j         @ Indicates the name of CounterVariable is j
  j +       @ Recall the value of j and adds to sum
 NEXT       @ Ends the loop
»
```

This program does not validate your input.

FOR – STEP

This loop has the following syntax:

InitialValue FinalValue

FOR

CounterVariable

[body of the loop]

StepValue

NEXT

There is no much to say here, everything that was said about the START-STEP loop applies here with the difference that now we have a CounterVariable that we can use in the loop.

Example 1: The Maclaurin series of sin(x) is:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \ldots$$

Given a real number as input, this program find the Maclaurin series of sin(x) truncated to the first six terms, we are not going to use any variables here, notice that we need a sum and a number one that is going to switch from positive to negative in each iteration, let's call it SignValue.

```
«
 1 0                 @ Sets SignValue and sum
 1. 11.              @ InitialValue and FinalValue are set as reals
 FOR
  j
  PICK3 j ^ j ! /    @ Calculate the j(th) termn, x was in level 3:
  PICK3              @ Recall SignValue witch was in level 3:
  * +                @ Multiply by j(th) termn and add to sum
  SWAP NEG SWAP      @ swaps sign_value change its sign and swaps it back
  2                  @ StepValue is constant and set to 2
 STEP
 NIP NIP             @ Clear x and sign_value
»
```

For 1 this program returns:

.841470984648

and the function SIN in the calculator returns:

.841470984808