1 - Three-Address Code (TAC)

We have the following VSL program:

```
func main()
begin
  var iter, count
  iter := 2
  while iter < 40 do
  begin
    count := collatz(iter)
    print iter, "converges in", count, "steps"
    iter := iter + 1
  end
  return 0
end

func collatz(n)
begin
  var steps
  steps := 0
  while n > 1 do
  begin
    var i
    i := n/2
    if n = i * 2 then n := i
    else n := 3 * n + 1
    steps := steps + 1
  end
  return steps
end
```

Translating this VSL program to three-address code (TAC) gives us the following:

```
# main
main:
  iter = 2
  goto L1
L0:
  param iter
  count = call collatz, 1

  param iter
  param "converges in"
  param count
  param "steps"
  call print, 4

  iter = iter + 1
L1:
  if iter < 40 goto L0
  return 0

# collatz
collatz:
  steps = 0
  goto L4
L2:
  i = n / 2
  if n = i * 2 then L6
  else goto L5
L3:
  steps = steps + 1
  goto L2
L4:
  if n > 1 goto L2
  return steps
L5:
  t1 = 3 * n
  t2 = t1 + 1
  n = t2
  goto L3
L6:
  n = i
  goto L3
```

In the TAC above, there are a few things to note:

- The line `count = call collatz, 1` describes a function call to `collatz` with one argument. The argument is passed via the `param` instruction on the line above. The `call` instruction returns the value of the function call in the variable `count`.
- The same goes for the `print` instruction. The arguments are passed via the `param` instructions on the lines above. The `call` instruction does not return a value.