Kompilatorteknikk TDT4205 - Problem set 4

Ola Kristoffer Hoff

14th March, 2023

1 Three-Address Code (TAC)

Translate the following VSL program into TAC. You may assume that printing can be done via a call to an external function with an argument signature of your own choosing. (*Feel free to make further assumptions about the execution environment if you find it necessary, but state them in your answer.*)

```
func main()
begin
    var iter, count
    iter:=2
    while iter < 40 do
    begin
        count:=collatz(iter)
        print iter, "coverges 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
```

Below you will find my translation to TAC. I assumed that when a function is called the parameters are already "loaded" with the values, so we don't have to pop them from the stack. I used a function-calling convention from the recitation lecture slides. For "print" I simply removed the return value for the syntax. Also, the parameters added will be read in the order they were added as left to right.

```
func collatz(n):
        steps = 0
.L3:
        if n > 1 goto .L4
        return steps
.L4:
        i = n / 2
        t1 = i * 2
        if n != t1 goto .L5
        n = i
        goto .L6
.L5:
        t2 = 3 * n
        n = t2 + 1
.L6:
        steps = steps + 1
        goto .L3

func main():
        iter = 2
.L1:
        if iter < 40 goto .L2
        return 0
.L2:
        param iter
        count = call collatz, 1
        param iter
        param "coverges in"
        param count
        param "steps"
        call print, 4
        iter = iter + 1
        goto .L1
```

2 Symbol Table Creation

I don't have much to comment on here. See the implementation in code.