Tail Recursion

Effective Programming in Scala

Reminder: Evaluating Recursive Factorial

```
def factorial(n: Int): Int =
  if n == 0 then 1
  else n * factorial(n - 1)
```

- factorial(3)
- 3 * factorial(3 - 1)
- 3 * 2 * factorial(2 - 1)
- 3 * 2 * 1 * factorial(1 - 1)
- 3 * 2 * 1 * 1
- 6

Reminder: Evaluating Recursive Factorial

```
def factorial(n: Int): Int =
  if n == 0 then 1
  else n * factorial(n - 1)
```

- `factorial(3)`
- `3 * factorial(3 - 1)`
- `3 * 2 * factorial(2 - 1)`
- `3 * 2 * 1 * factorial(1 - 1)`
- `3 * 2 * 1 * 1`
- `6`

Each time the runtime evaluates a `factorial` call, it pushes its parameter to the **call stack**.

Overflowing the Call Stack

If the chain of recursive calls is too long the call stack overflows, which produces a runtime error called a `StackOverflowError`.

The maximal number of iterations before getting a `StackOverflowError` depends on the runtime, it is in general around tens of thousands.

Overflowing the Call Stack

If the chain of recursive calls is too long the call stack overflows, which produces a runtime error called a `StackOverflowError`.

The maximal number of iterations before getting a `StackOverflowError` depends on the runtime, it is in general around tens of thousands.

Luckily, it is possible to not use stack space by putting the recursive call in *tail* position.

Tail Recursive Factorial

A recursive call is in tail position if it is the result of the recursive method (ie, there is no further operation applied to it).

```
def factorial(n: Int): Int =
  def factorialTailRec(x: Int, accumulator: Int): Int =
    if x == 0 then accumulator
    else factorialTailRec(x - 1, x * accumulator)

  factorialTailRec(n, 1)
end factorial
```

With this definition, calls to `factorialTailRec` don't need to use space on the call stack.

Tail Recursive Factorial Call

```
def factorialTailRec(x: Int, accumulator: Int): Int =
  else if x == 0 then accumulator
  else factorialTailRec(x - 1, x * accumulator)
```

- factorialTailRec(3, 1)
- factorialTailRec(3 - 1, 3 * 1)
- factorialTailRec(2, 3)
- factorialTailRec(2 - 1, 2 * 3)
- factorialTailRec(1, 6)
- factorialTailRec(1 - 1, 1 * 6)
- factorialTailRec(0, 6)
- 6

Summary

The call stack size may limit the number of possible iterations of a recursive method, unless the recursive call is in tail position.