Tail Recursion

Why do we need tail recursion?

Consider a function for the factorial operation:

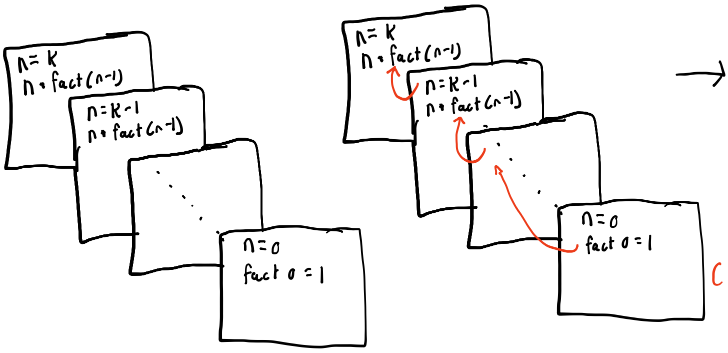
```
let rec fact n =
  if n=0 then 1
  else n * fact (n-1)
```

Try: fact 0 | fact 15 | fact 1,000,000,000

stack overflow

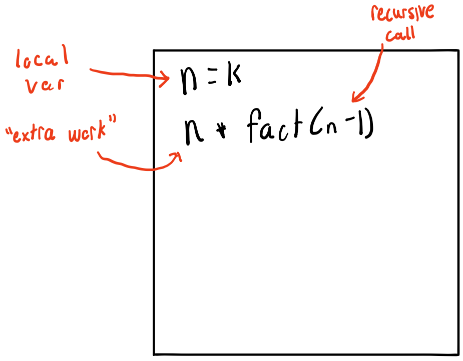
O(n) space

The call stack:



(completed and popped)

→ Depth of call stack is limited (for safety)!



Notice:
call stack contains one element for each function call that is started but not completed yet

the "extra work" is the only reason we still need this element of the call stack

If we can find a way to write functions with no extra work, is there some trick to reducing space complexity of recursive functions?

Tail Call Optimization

- An agreement between programmer and compiler

Programmer:

- Write good recursive programs with recursive calls in "tail position" (leaving no extra work)

Compiler:

- Figure out when a function is tail recursive and reuse old call frames to achieve O(1) space

Trust the compiler!

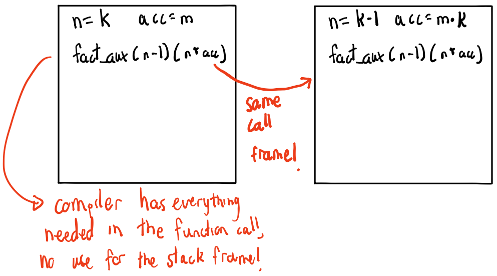
How do we do our job?

Idea: Do "extra work" inside a function parameter, accumulating the final answer over time. still need to maintain the base case.

```
let rec fact_aux n acc =
  if n=0 then acc
  else fact_aux (n-1) (n * acc)

let fact n =
  fact_aux n 1
```

no additional computation after recursive call!



A Method for Tail Recursion

1. Turn recursive function into helper function (aux). Add the param for accumulation.
2. Write new main function that calls helper with base case as acc. Same sig as original.
3. Change helper to return acc in its base case
4. Modify helper recursive case to do "extra work" within the acc parameter, before recursive call is made. This is the thinking part.

Original:

```
let rec fact n =
  if n=0 then 1
  else n * fact (n-1)
```

① →

* intermediate step unused var

```
*let rec fact_aux n acc =
  if n=0 then 1
  else n * fact_aux (n-1)
```

③ →

```
*let rec fact_aux n acc =
  if n=0 then acc
  else n * fact_aux (n-1)
```

④

```
let rec fact_aux n acc =
  if n=0 then acc
  else fact_aux (n-1) (n * acc)
```

②

```
let fact n =
  fact_aux n ①
```

Quiz Questions

1. Does tail recursion change time complexity?
2. Can all recursive functions be made tail recursive?