Subprograms - Part 2
Recursion

CS 2XA3

Term I, 2018/19

Outline

Recursion

- ***Direct recursion***: a function calls itself
- ***Indirect recursion***: function A_1 calls function A_2, A_2 calls function A_3, ..., A_{n-1} calls A_n, and then A_n calls A_1.

Example: computing $n!$

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n(n-1)! & \text{if } n > 1 \end{cases}$$

In Python:

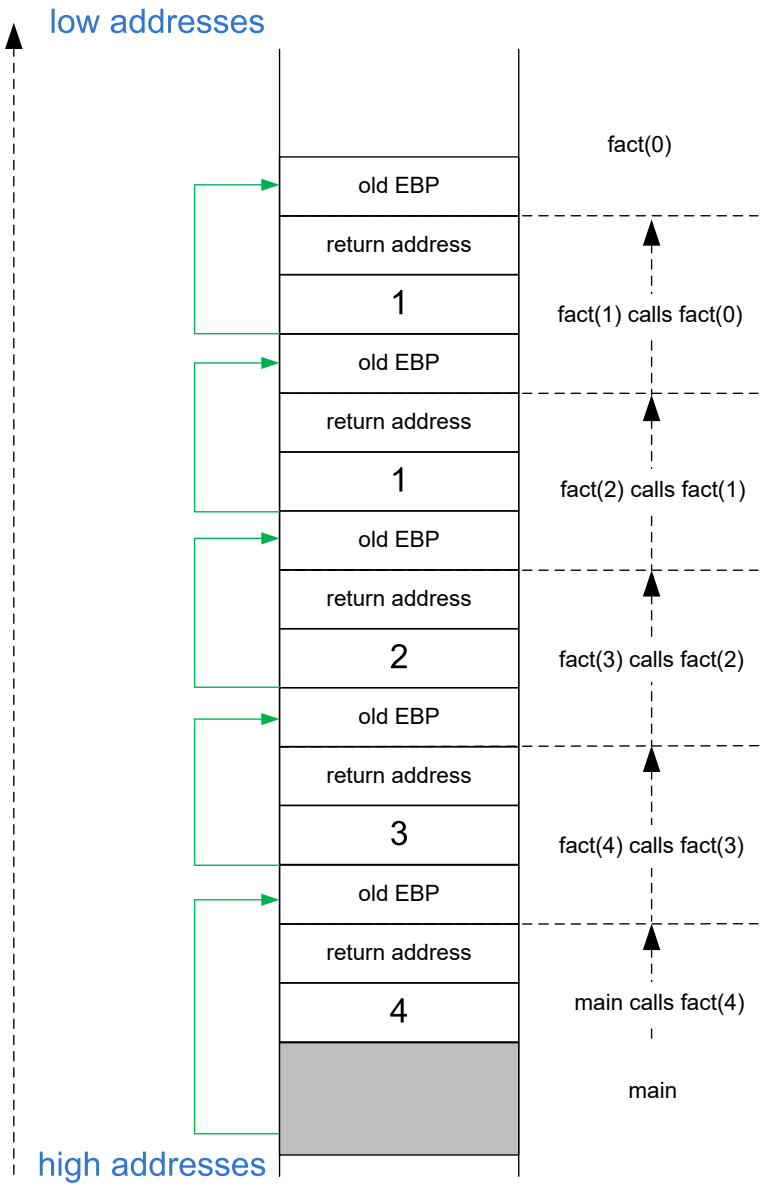
```
def fact(n):
    if n==0:
        return 1
    else
        return n*fact(n-1)
```

NASM code for `fact( )`

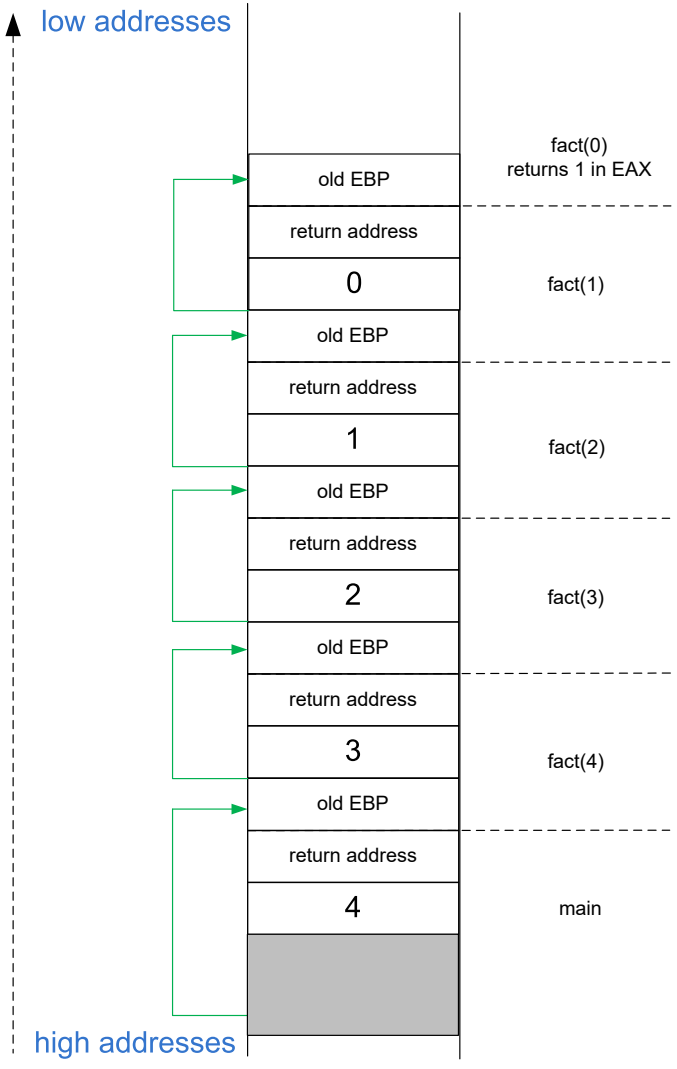
```
section .text
global fact:
    enter 0,0
    mov eax, [ebp+8]                ;eax=8
    cmp eax, 0                      ;if n==0
    je term_cond
    dec eax                         ;n=n-1
    push eax
    call fact                       ;fact(n-1)
    pop ecx                         ;eax=n
    mul dword [ebp+8]               ;eax=n*fact(n-1)
    jmp end_fact
term_cond:
    mov eax, 1
end_fact:
leave
ret
```

- Assume a program calls `fact(4)`
  - it pushes 4 onto the stack
  - calls `fact(4)`
- `fact(4)`
  - saves old `ebp`
  - sets `ebp` to `esp`
  - sets `eax` to 4
  - pushes 3 onto the stack
  - calls `fact(3)`
    - `fact(3) computes 6 and stores it into eax`
  - pops the parameter 3 into `ecx`
  - multiplies `[ebp+8]` by `eax`
  - stores the result 24 in `eax`
  - clears the stack

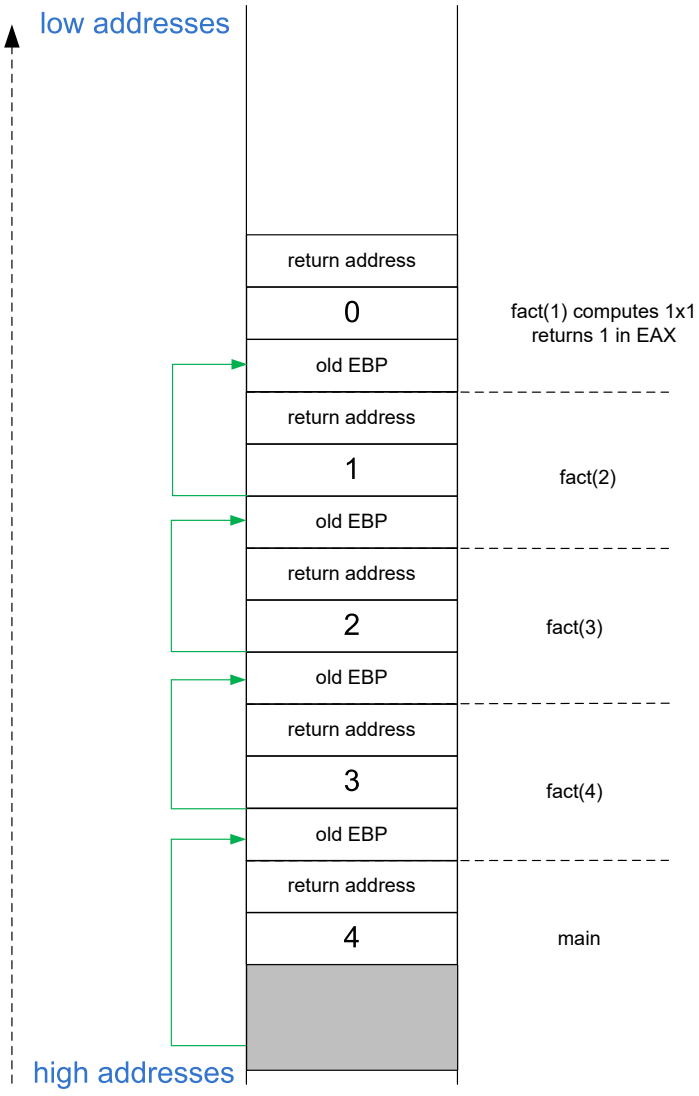
The stack after `fact(0)` is called



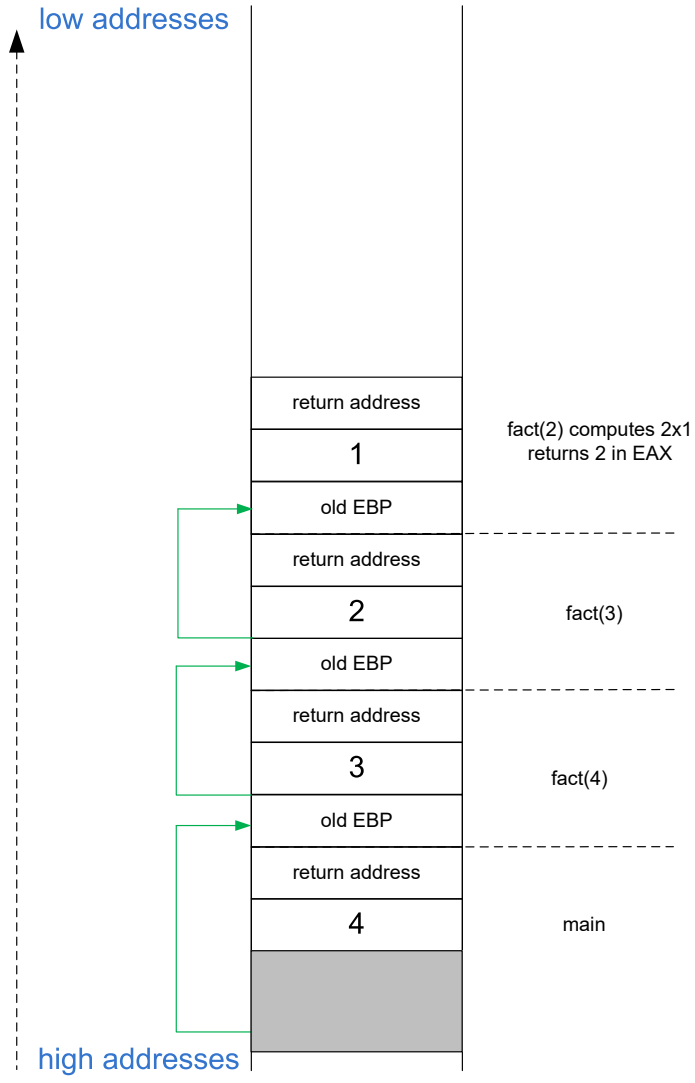
`fact(0)` stores 1 in `eax` and returns `fact(1)`



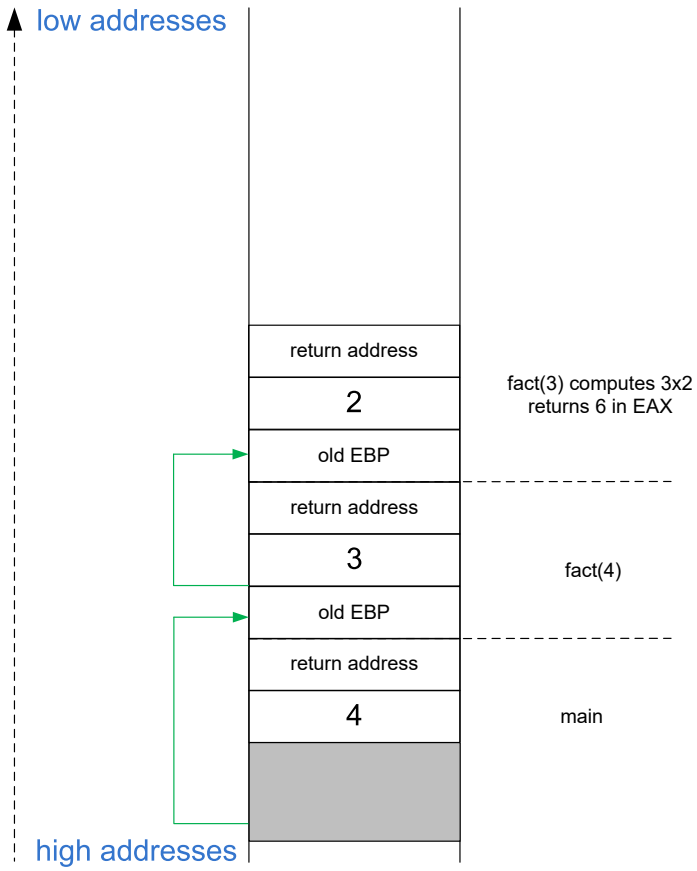
The stack after `fact(0)` is done. `fact(1)` computes `1 x 1`, stores `1` in `eax` and returns `to fact(2)`



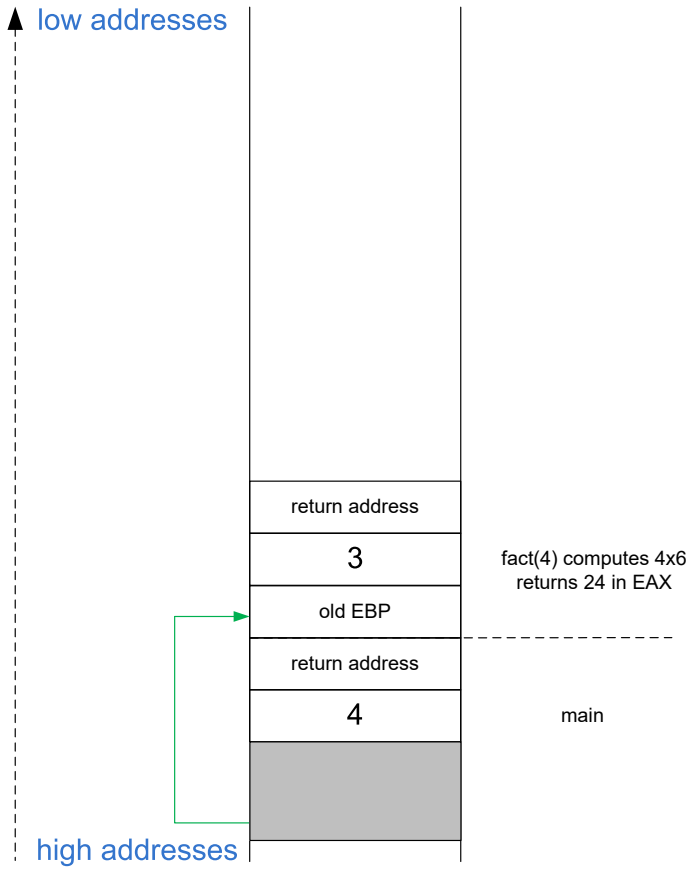
The stack after `fact(1)` is done. `fact(2)` computes `2 x 1`, stores `2` in `eax` and returns to `fact(3)`



The stack after `fac(2)` is done. `fac(3)` computes `3 x 2`, stores `6` in `eax` and returns to `fact(4)`



The stack after `fac(3)` is done. `fac(4)` computes `4 x 6`, stores `24` in `eax` and returns to `main`



The stack after `fac(4)` is done. `main` continues its execution by “cleaning” the stack.