Recursive subprograms

Ned Nedialkov

McMaster University
Canada

February 2015

Outline

Recursion

- Direct recursion: a function calls itself
- Indirect recursion: function A calls function B and then B calls A

Example: computing n!

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

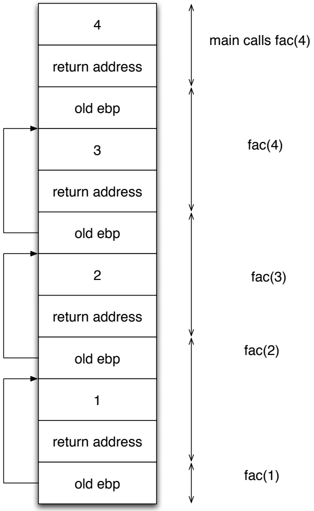
```
int fac(int n)
{
  if (n==1) return 1;
  return n*fac(n-1);
}
```

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

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

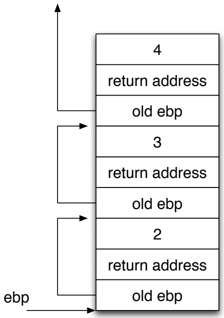
© 2013–15 Ned Nedialkov

The stack after `fac(1)` is called



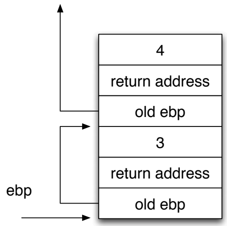
The stack after `fac(1)` is done

`fac(2)` computes 1×2, stores 2 in **eax** and returns to `fac(3)`



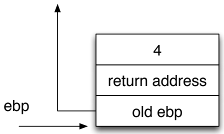
The stack after `fac(2)` is done

`fac(3)` computes 2×3, stores 6 in **eax** and returns to `fac(4)`



The stack after `fac(3)` is done

`fac(4)` computes 6×4, stores 24 in **eax** and returns to the main program


4
return address
old ebp
ebp