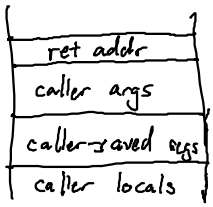
C Calling Conventions on x86-64 under Linux

caller-saved / volatile: caller must save if they matter

1. Caller setup and call
   a. Push caller-saved regs
   b. Set up arguments — store them in param locs
      rdi, rsi, rdx, rcx, r8, r9
      rest of args go on stack in "reverse" order (in memory order)
   c. Issue a "call" instruction

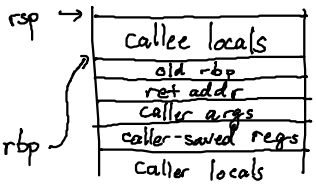


2. Callee sets up stack frame
   a. push rbp
   b. mov rbp, rsp
   c. sub rsp, #

call foo ≡ push rip*
jmp foo

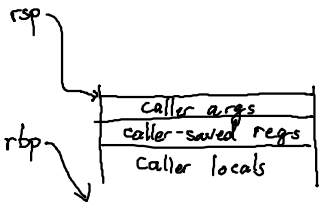
push rax ≡ sub rsp, 8
mov [rsp], rax

3. DO STUFF
   result in rax



4. Callee tears down stack frame
   a. mov rsp, rbp
   b. pop rbp
   c. ret

ret ≡ pop rip



5. Caller tear down stack from call
   a. Remove any args from stack
   b. Restore any caller-saved regs

As a C compiler: generate assembly to call a function named "foo" with args 5, 7 — but I need to save r8

answer = f(5,7)

```
push r8
mov rdi, 5
mov rsi, 7
call foo
pop r8
```

Cardinal must call a C function: Cardinal must be a C caller

Cardinal must be called by the driver: Cardinal must be a C callee

Address stack memory as e.g. [rbp-8] instead [rsp-8]

```
let n = 2 in
let m = after (n) in
m
```

```
bird_main:
step 2   push rbp
         mov  rbp, rsp
         sub  rsp, 16
step 3   mov  rax, 4
         mov  [rbp-8], rax
         mov  rax, [rbp-8]
         add  rax, 2
         mov  [rbp-16], rax
         mov  rax, [rbp-16]
step 4   mov  rsp, rbp
         pop  rbp
         ret
```

Dove

<program> ::= <decl_list> <expr>

<decl_list> ::= ϵ | <decl> <decl_list>

<decl> ::= def <ident> (<param_list>) <expr> end

<param_list> ::= ϵ | <param> | <param>, <param_list>

<param> ::= <ident>

<expr> ::= .... | <ident> (<expr_list>)

```
def triple(n)
  n+n+n
end
triple(triple(4))
```

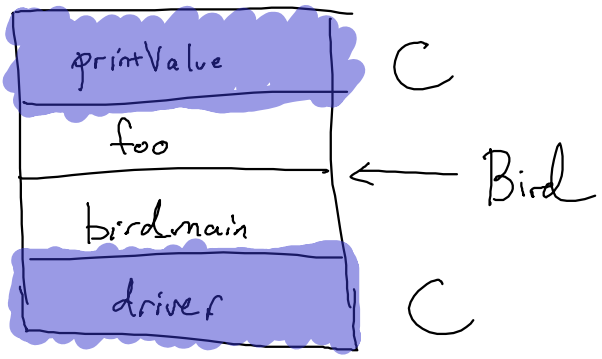
⟹ 36

```
def f(a)
  a+2
end
f(4)
```

⟹ 6

Bird Calling Conventions

Same as C calling conventions except all arguments are stored in stack memory




```
def f(a)
  a+2
end
f(4)
```

⟹

```
fn_f:
    push rbp
    .
    .
    .
    pop rbp
    ret
bird_main:
    push rbp
    .
    .
    .
    call fn_f
    .
    .
    .
    pop rbp
    ret
```