Procedures & Environments

declaration void f(int);

definition void f(int x){
....
}

activation f(x);

push args on stack or load registers
call f
rescue returned result (if any)

function:
allocate local stack frame
execute insns
load return value (if any)
load return address
pop local stack frame
return

environment:
set of global and non-local variables available
global - not nested in a block
non local - nested in an outer block

function call stack:
stack of local stack frames

C scopes:
- static extern: available everywhere
- static file scope: limited to all fns one file
- static function: limited to fn
- automatic: alloc @ entry; pop @ return

fns may not be nested
but scopes inside fns may be
C++: more access protection (pri; prot; pub)

Parameter Passing

By value (copy)
- argument is copied
- copy ctor if object (might be expensive)
- Java: only primitives

By value (move)
- C++ object is moved into fn
- no longer available to caller

By reference
- pass address of or reference to object
- Java: all objects
- more efficient
- C++ reference or const reference

By Value-Result (copy in, copy back)

- copy by value in
- do function
- copy back @ return possibly changed var
- Ada.

By Result (copy back)

- param is uninitialized local
- copy back to caller @ return

By Name (thunk)

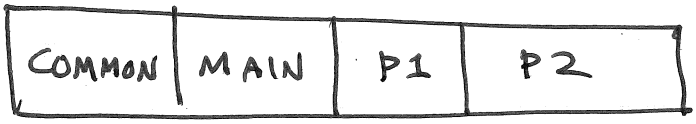
- pass in address of a thunk,
  i.e. an unevaluated lambda
  with no args
  Scheme: (f (lambda () (....))
- default in Haskell
- C++11: f ([](int) -> int {...})
  ↑ lambda
- some thunks (lambdas) may have params
- delayed evaluation

Environment, Activation, Allocation

- nonlocal refs
- static refs

Fully Static (FORTRAN IV)

- no stack
- all variables static
- recursion not possible

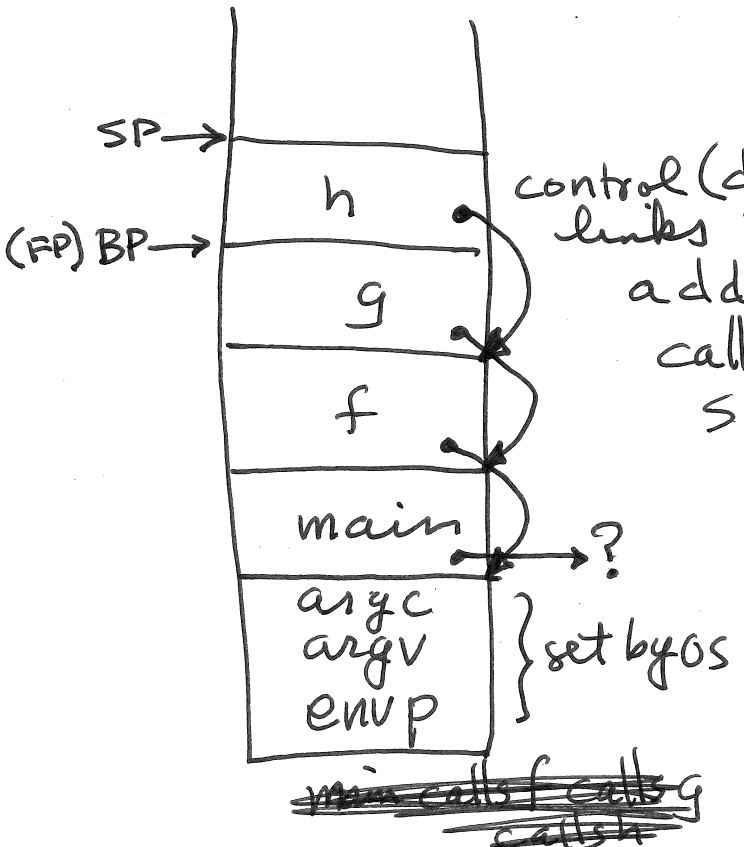


- ex: call on Univac 1100

call: SLJ F
stores a ret jmp

F: RES 1
≡
JMP F

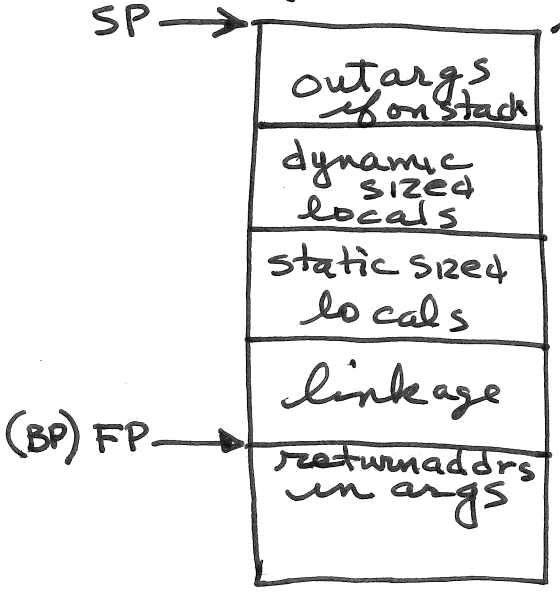
Stack based runtime (C) no nested fns




```
h(){
}
g(){
   h()
}
f(){
   g()
}
main(){
   f()
}
```

activation record (local stack frame)

NO NESTING C C++ Java

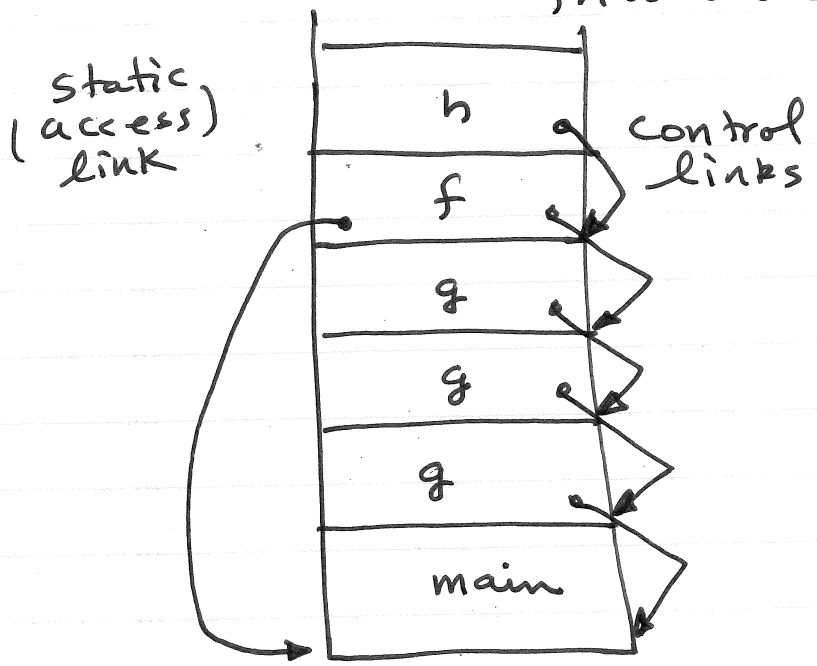


1. caller: push args on stack or load regs
   call: pushes ret addr on stack or load reg
2. establish local frame
   push FP (caller's FP)
   FP = SP (callers ceiling is our floor)
   SP -= alloc (alloc frame)
3. save regs in linkage

⋮

n. return: load return value (usually a register)
   SP = FP
   pop FP (caller's FP)
   return

* Nested fns activation record
  control (dynamic) link
  → caller's activation record
  access (static) link
  → stackframe (act. rec.) of fn inside which we are nested




```
g() {
  calls g(f); calls f
}
main() {
    f() { h() }
    g(f)
}
```

access (static) link contains addr of stack frame inside which fn is nested
problem if it is no longer on stack
--- dangling pointer
fn addrs is then pair

code addr	access link

- need to chain down access links if multiple nesting
- problem: usually not nesting is usually shallow

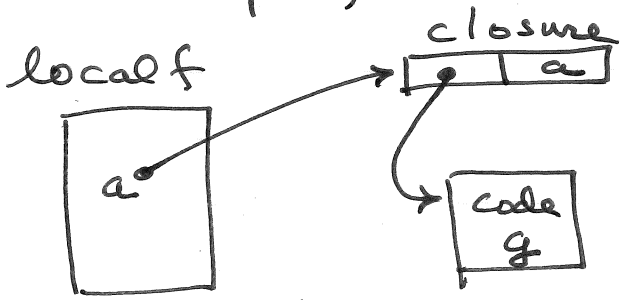
What if nested fn is returned?
then access link is dangling

∴ use closure to hold escaping vars
— problem in func lang,
not in no-nest imper lang

Closure : heap allocated structure

```
f() {
  int a;
  g() {
    use a
  }
  return g
}
```

a escapes f.



pass address of closure
to call g
if p → g (*p)(p)

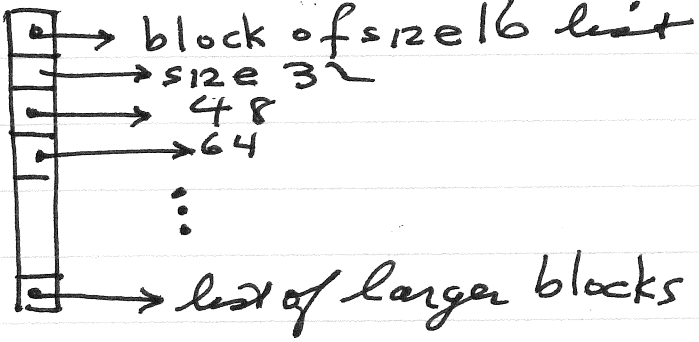
like a VFT with one fn.

Dynamic Memory Managgnt (HEAP)

allocate : malloc/calloc/realloc/new

free : free / destructor / gcol

maintain a free list



Reclamation

free – put back on list

destructor – auto calls free

* reference count
  - keep a count of refs
  - incr on copy
  - decr on assign
  - free when 0
* mark & sweep
  - mark closure(root set)
  - free all unmarked
* copying
  - copy all live obj to new heap
  - mass free old heap
* generational