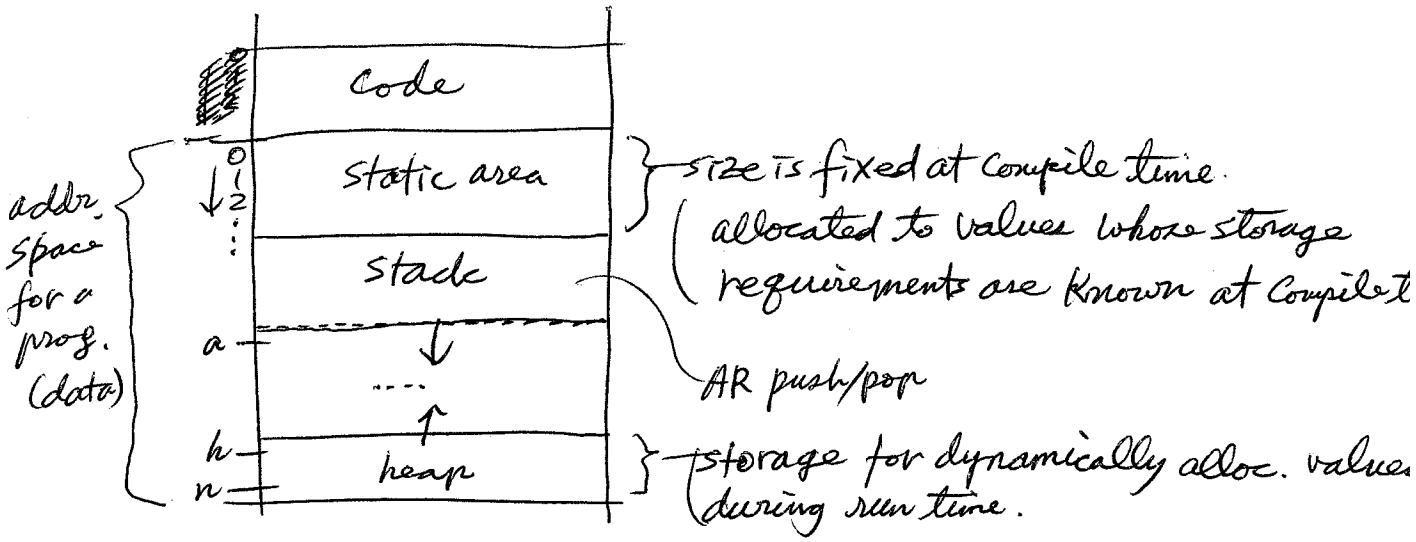
Ch10. Implementing Subprograms

Parke note

—Run-time memory management



values → memory
assigned to
static area
stack
heap
} dynamic mem.
needs garbage collection

—invariant for preventing stack-overflow (at any time during ru

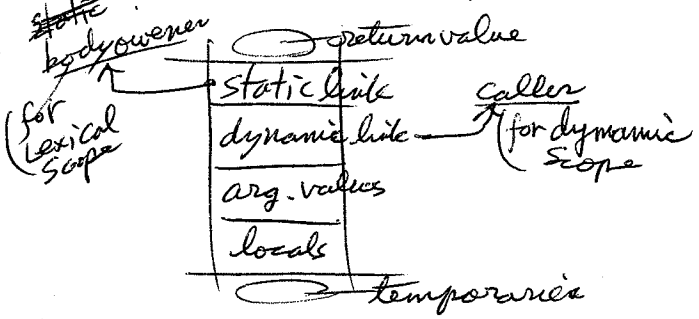
$$0 \le a \le h < n$$

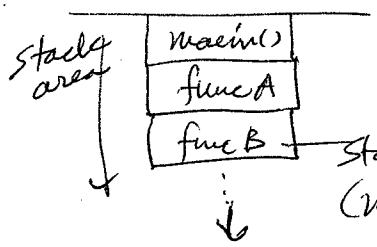
stack top
(initially defined at the beginning of run time) } depending upon prog.

—max. stack size — varies among different prog's
based on { nesting-level
recursion-level

— A.R. and runtime stack

— Activation record for a func invocation



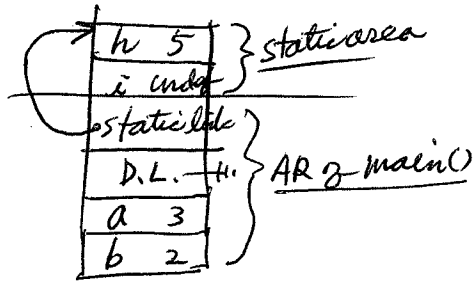


— Java like syntax (static scope)

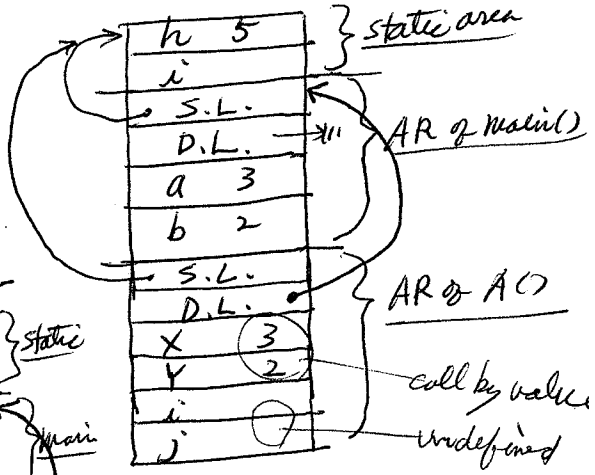
```
Package K
{ int h, i;                          globals (static area)
  void A (int x, int y)
  { boolean i, j;
    B(h);
    :
  }
  void B (int w)
  { int j, k;
    i = 2*w;                         i in static scope, i is global, use static link
    w = w+1;                         i in dynamic scope, i is A's i, use dynamic link
    :
  }
  void main()
  { int a, b;
    h = 5;
    a = 3;
    b = 2;
    A(a, b);
    :
  }
}
```

Java / JSP / PL

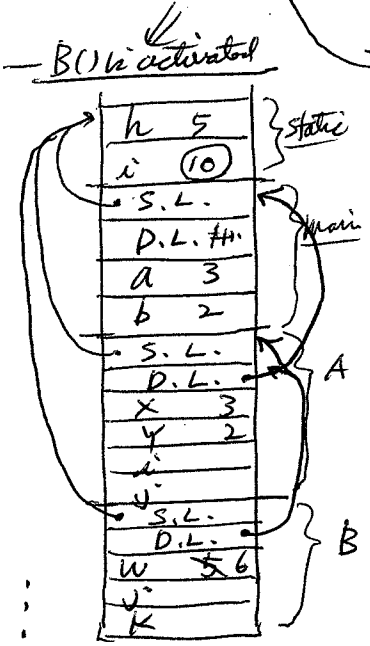
— main() is activated



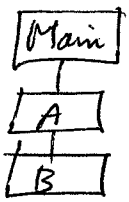
— A() is activated



— B() is activated



— Activation tree



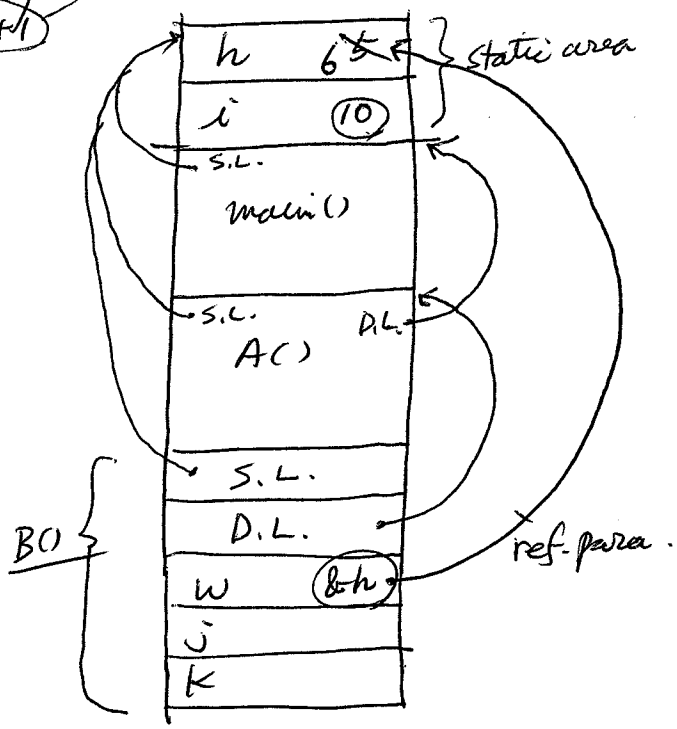
++ call by reference case — C/Java — only call-by-value.

— C++ like syntax

```
void B (int& w)
{ int j, k;
  i = 2*w ;  ⇒ (i=2*h) effect
  w = w+1 ;  ⇒ (h=h+1)
  :
}
```

&h (addr. of h) global

— When B() is activated,



↓ main() calls A();
A() calls B(h);

— C/C++/Java
(≠ nested subprogram definition
⇒ S.L. (static link always points to the static area.

— caller sets S.L./D.L. and parameters, and return addr.
in the AR of callee.
then, callee sets remaining fields (locals)

— recursion/nested subprograms

name occurence → declaration → location → value

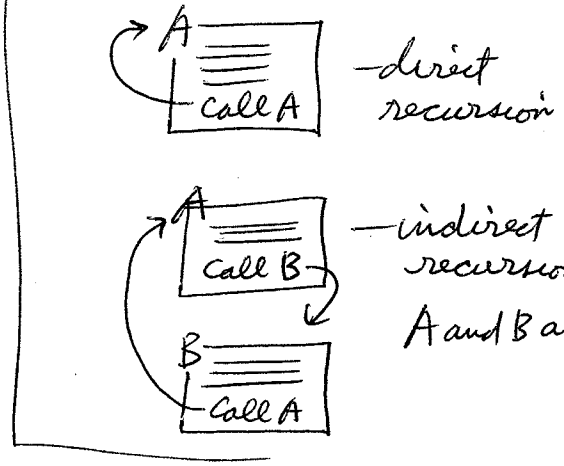
(activation time — (run time

ex) assign-statement

(locals in recursive subprogram activation denote different locations

A.R.'s

ref. recursion



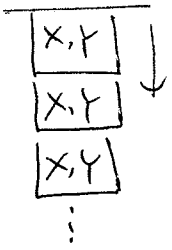
— direct recursion

— indirect recursion

A and B are involved in a recursion.

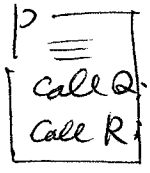
ex) func A

```
{ var X, Y;
  :
  X=1;
  Call A;
}
```



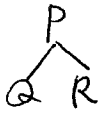
— Activation tree (shows all caller/callee relationship during the execution of the program.

ex)

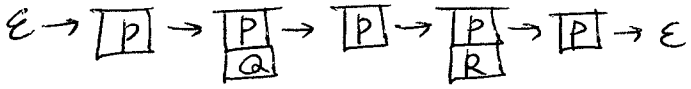


—

Activation tree

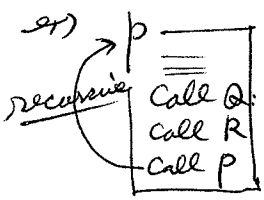


run-time stack (Snapshots)



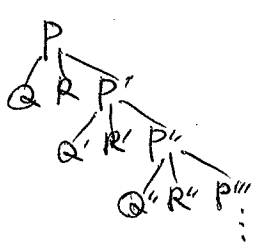
---

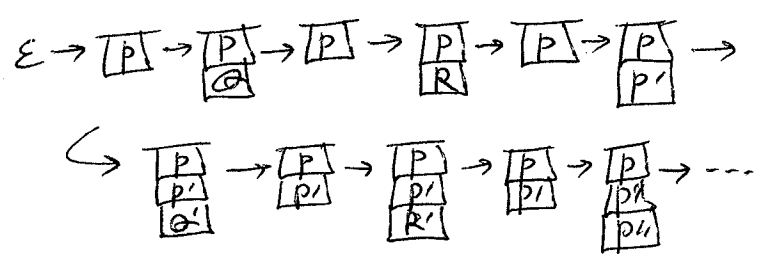
ex)



(never ending)

—

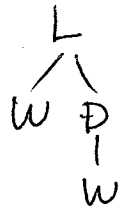




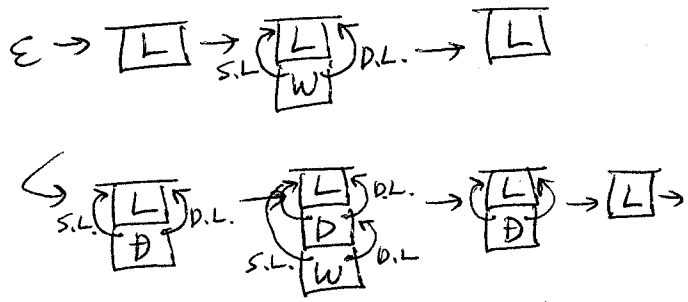
4) — nested static scope lang. (e.g., PASCAL)

```
program L ;
  or procedure
  var n : char ;
  proc W ;
    begin
      writeln (n) ;
    end
  proc D ;
    var n : char ;
    begin
      n := 'D' ;
      W ;
      D ;
    end ;
  begin
    n := 'L' ;
    W ;
    D ;
  end .
```

— Activation tree



— run-time stack



program has globals

vs.

C/C++, main() has locals

non-nested static scope

S.L. points to static area.

always

⇒ in fact, S.L. is not needed.

static link (access link)
for static scope

dynamic link (control link)
for accessing caller's environment

— Static link (access link)

points to the environment of definition of the proc/func.

for nested static scope,

1. Compute the static distance (nesting levels) between caller/callee:

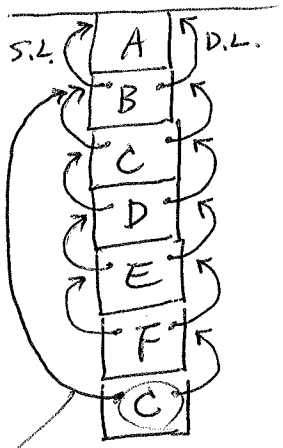
   d = nesting_level_caller − nesting_level_callee + 1

   (nesting levels are kept in symbol table)

2. start from the caller's AR, trace S.L. (d) times, and set the callee's S.L. to that environment.

4)

```
proc A (nesting level 1)
    proc B (nesting level 2)
        proc C (nesting level 3)
            proc D (nesting level 4)
                proc E (nesting level 5)
                    proc F (nesting level 6)
                        → call C();
                    → call F();
                → call E();
            → call D();
        → call C();
    → call B();
```



$d = 6 - 3 + 1 = 4$

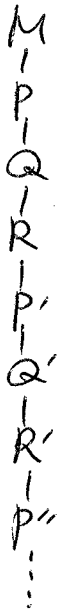
start from F's AR, trace S.L. 4 times.

Nested static scope

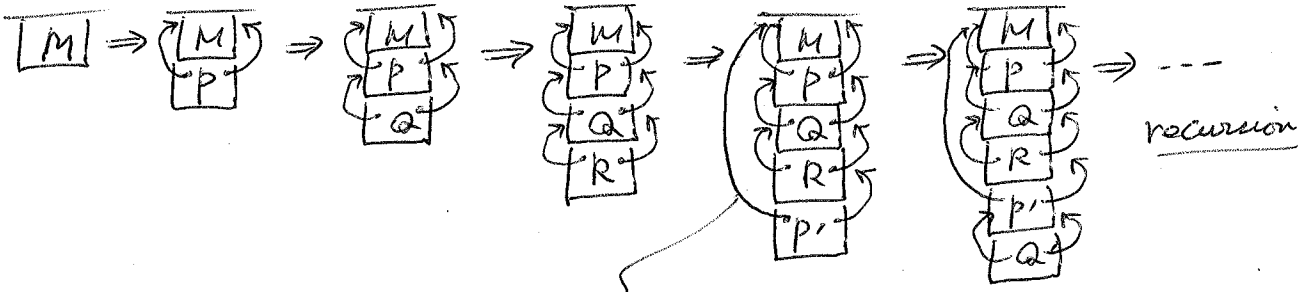
eg)

```
1 — proc_M:
    2 proc_P:
        var X, Y, Z
        3 proc_Q:
            4 proc_R:
                Z := (P)
            Y := (R)
        X := (Q);
    (P):
```

activation tree



run time stack



$d = 4 - 2 + 1 = 3$

trace from R's AR, S.L. 3 times.