CSc 520

Principles of Programming Languages

33: Procedures — Scope

Christian Collberg

collberg@cs.arizona.edu

Department of Computer Science

University of Arizona

Nested Subroutines

- Algol 60, Pascal, Ada, Modula-2, etc. allow procedures to be nested inside each other.
- Closest nested scope rule:
  - a name that is introduced in a declaration is known in the scope in which it is declared, and in each internally nested scope, unless it is hidden by another declaration of the same name.
  - To search for the declaration corresponding to a use of a name, we search outward from the current scope.
- Nested subroutines are able to access the parameters and local variables of surrounding scopes.

Nested Subroutines...

```
procedure P1 (A1:T1);
var X : real;
   procedure P2 (A2: T3);
      procedure P3 (A3 : T3);
      begin (* body of P3 *) end;
   begin
      (* body of P2 *)
   end;
   procedure P4 (A4: T4);
      function F1 (A5 : T5);
      var X : integer;
      begin (* body of F1 *) end;
   begin
      (* body of P4 *)
   end;
begin
   (* body of P1 *)
end;
```

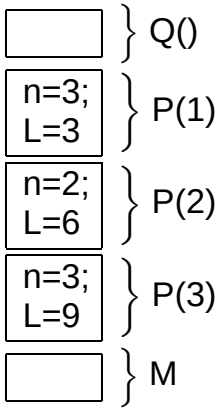

Accessing Non-Local Variables

```
PROGRAM M;
   PROC P(n);
      LOCAL L;
      PROC Q(); BEGIN PRINT L; END Q;
   BEGIN
      L := n * 3;
      IF n >= 1 THEN P(n-1) ELSE Q() ENDIF;
   END P;
```

- Which `L` should `Q` print? There are three `L`s on the stack to choose from!

Accessing Non-Local Variables

```
BEGIN P(3); END M. PROGRAM M;
  PROC P(n);
  LOCAL L;
    PROC Q();
    BEGIN PRINT L; END Q;
  BEGIN
    L := n * 3;
    IF n >= 1
      THEN P(n-1);
      ELSE Q();
    ENDIF;
  END P;
BEGIN P(3); END M.
```



- `Q` should print the `L` from the topmost `P` on the stack.

Accessing Non-Local Variables...

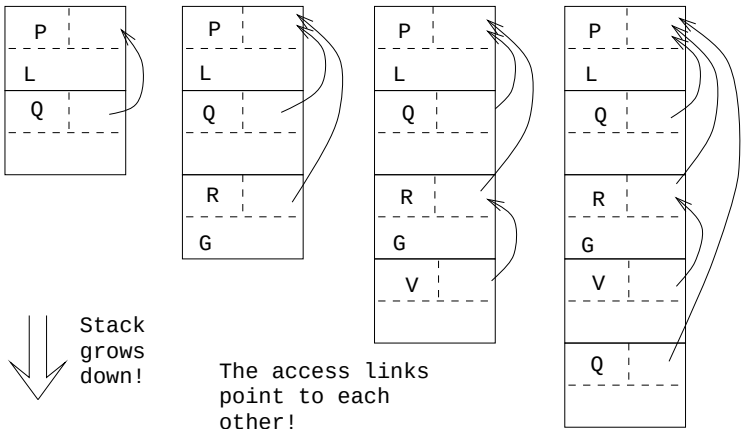
```
PROCEDURE P (a:INTEGER);
    VAR L : INTEGER:
    PROCEDURE Q (x:INTEGER);
    BEGIN R(16) END Q;

    PROCEDURE R (y:INTEGER);
        VAR G : INTEGER:
        PROCEDURE V (z:INTEGER);
        BEGIN Q(10) END V;
    BEGIN V(12) END R;

BEGIN Q (5); END P;
```


Accessing Non-Local Variables...

```
PROCEDURE P (a:INTEGER);
    PROCEDURE Q (x:INTEGER);
    PROCEDURE R (y:INTEGER);
        PROCEDURE V (z:INTEGER);
```

- We give each activation record an **Access Link** (aka **Static Link**).
- Assume that `Q` is nested within `P` (as above). Then `Q`'s static link points to the activation record for the most recent activation of `P`.

Accessing Non-Local Variables...



Accessing Non-Local Variables...

```
PROC P ();
    VAR L:INTEGER;                    ⇐ n_L = 1
    PROC R ();
        PROC V ();                    n_R − n_L = 2
        BEGIN L:=...END V;            ⇐ n_R = 3
```

Access to non-local variable L:

- Assume that L is declared at nesting level n_L, and that the reference to L is at nesting level n_R (as above).
- Follow $n_R - n_L$ access links. We now point to the activation record for the most recent activation of P.

Accessing Non-Local Variables...

```
PROC P ();
    VAR L:INTEGER;                    ⇐ n_L = 1
    PROC R ();
        PROC V ();                    n_R − n_L = 2
        BEGIN L:=...END V;            ⇐ n_R = 3
```

MIPS Example:

```
lw $2, AL($fp) # AL is offset of access link.
lw $2, ($2)    # An access link points to
               # the previous access link.
lw $3, 12($2)  # Get the data in the AR.
```

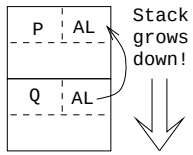

Setting up Access Links

- Every time we make a procedure call we have to set up the access link for the new procedure activation.
- There are two cases to consider:
  1. when the callee is nested within the caller, and
  2. when the caller is nested within the callee.

Setting up Access Links...

Case (1): Callee Within Caller:

```
PROC P();          ⇐ N_P = 1
    PROC Q();      ⇐ N_Q = 2
        PROC V();
        ...
BEGIN Q (); END P;
```

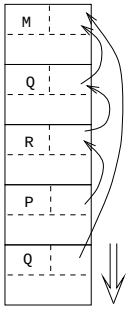


- P calls Q. P is at level N_P, Q is at level N_Q. $N_P = N_Q - 1$, since Q must be nested immediately within P.
- Make Q's access link point to the access link in P's activation record.

Setting up Access Links...

Case (2): Caller Within Callee:

```
PROG M;
  PROC Q();          ⇐ N_Q = 1
    PROC R();
      PROC P();      ⇐ N_P = 3
      BEGIN
        Q();
      END P;

      N_P − N_Q + 1 = 3
```



- P calls Q. P is at level N_P, Q is at level N_Q. $N_P \geq N_Q$.
- Traverse the access links to find the most recent activation of the first procedure which statically encloses both P and Q. We need to follow $N_P - N_Q + 1$ links.

Readings and References

- Read Scott, pp. 115–121, 427–433