Parameter Passing – Reference Parameters

PROG M;
PROC P(VAR X:INT);
BEGIN
  X:=5
END P;
VAR S:INT;
BEGIN
  S:=6;
P(S);
END.

- Reference parameters are passed by passing the address (location, l-value) to the parameter. Changes to a formal affects the actual also.

Call-by-Value Parameters

1. The caller computes the arguments’ r-value.
2. The caller places the r-values in the callee’s activation record.
   - The caller’s actuals are never affected by the call.
   - Copies may have to be made of large structures.

TYPE T = ARRAY 10000 OF CHAR;
PROC P (a:INTEGER; b:T);
BEGIN
  a:=10;
b[5]:="4"
END P;

VAR r : INTEGER; X : T;
BEGIN
  P(r, X)
END

Parameter Passing – Value Parameters

PROG M;
PROC P(X:INT);
BEGIN
  X:=5
END P;
VAR S:INT;
BEGIN
  S:=6;
P(S);
END.

- Value parameters are (usually) copied by the caller into the callee’s activation record. Changes to a formal won’t affect the actual.
### Call-by-Reference Parameters

1. The caller computes the arguments' *l-value*.
2. Expression actuals (like `a + b`) are stored in a new location.
3. The caller places the *l-values* in the *callee's* activation record.
   - The caller's actuals may be affected by the call.

```plaintext
TYPE T = ARRAY 10000 OF CHAR;
PROC P (VAR a:INT; VAR b:T);
BEGIN a:=10; b[5] := "4" END P;
```

```plaintext
VAR r : INTEGER; X : T;
BEGIN P(5 + r, X) END
```

### Call-by-Name Parameters

1. The caller passes a *thunk*, a function which computes the argument's *l-value* and/or *r-value*, to the callee.
2. The caller also passes a static link to its environment.
3. Every time the callee references the name parameter, the thunk is called to evaluate it. The static link is passed to the thunk.

### Algorithm:

1. If the parameter is used as an l-value, the thunk should return an l-value, otherwise an r-value.
2. If the parameter is used as an l-value, but the actual parameter has no l-value (it's a constant), the thunk should produce an error.

### Consequences:

- Every time a callee references a name parameter, it may produce a different result.

```plaintext
VAR i : INTEGER; VAR a : ARRAY 2 OF INTEGER;
PROCEDURE P (NAME x:INTEGER);
BEGIN
  i := i + 1; x := x + 1;
END;
```

```plaintext
BEGIN
  i := 1; a[1] := 1; a[2] := 2;
P(a[i]);
  WRITE a[1], a[2];
END
```

- `x := x + 1` becomes `a[i] := a[i] + 1`.
  ⇒ Print 1,3.

### Call-by-Name Parameters...
Large Value Parameters

- Large value parameters have to be treated specially, so that a change to the formal won’t affect the actual. Example:

```pascal
TYPE T = ARRAY [1..1000] OF CHAR;
PROCEDURE P (x : T);
BEGIN
  x[5] := "f";
END P;
VAR L : T;
BEGIN
  P(L);
END.
```

Call-by-Name Parameters – Implementation

```pascal
PROCEDURE P (thunk : PROC());
BEGIN
  i := i + 1; thunk()↑ := thunk()↑ + 1;
END;

PROCEDURE thunk1 () : ADDRESS;
BEGIN RETURN ADDR(a[i]) END;
BEGIN
  i := 1; a[1] := 1; a[2] := 2;
  P(thunk1);
  WRITE a[1], a[2];
END
```

Large Value Parameters...

```pascal
Algorithm 1: Callee Copy
PROCEDURE P (VAR x : T);
VAR xT : T;
BEGIN
  copy(xT,x,1000);
  xT[5] := "f";
END P;
VAR L : T;
BEGIN
  P(L);
END
```

```pascal
Algorithm 2: Caller Copy
PROCEDURE P (VAR x : T);
VAR xT : T;
BEGIN
  x[5] := "f";
END P;
VAR L : T;
BEGIN
  copy(LT, L, 1000);
  P(LT);
END
```

Call-by-Name – Jensen’s Device

```pascal
PROC Sum (NAME Expr:REAL; NAME Idx:INTEGER; Max:INTEGER):INTEGER;
VAR Result : REAL := 0;
BEGIN
  FOR i := 1 TO Max DO;
    Idx := i; Result := Result + Expr;
  END;
  RETURN Result;
END;

VAR i : INTEGER;
BEGIN
  WRITE Sum(i, i, 5); (* \( \sum_{i=1}^{5} i \) *)
  WRITE Sum(i*i, i, 10); (* \( \sum_{i=1}^{10} i^2 \) *)
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 Ls on the stack to choose from!

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 V(10) END V;
BEGIN V(12) END R;
BEGIN Q (5); END P;

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...

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(1)
  n=3; L=3

  P(2)
  n=2; L=6

  P(3)
  n=3; L=9

END P;
BEGIN P(3); END M.

Q should print the L from the topmost P on the stack.
Accessing Non-Local Variables...

PROC P ();
VAR L:INTEGER; \( \Leftarrow n_L = 1 \)
PROC R ();
PROC V (); \( n_R - n_L = 2 \)
BEGIN L:=...END V; \( \Leftarrow n_R = 3 \)

MIPS Example: 

\[ \text{lw } \$2, \text{AL}($fp) \] # AL is offset of access link.
\[ \text{lw } \$2, ($2) \] # An access link points to
    # the previous access link.
\[ \text{lw } \$3, 12($2) \] # 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.

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.
### Parameter Passing

- In Pascal, parameters are passed either by value or by reference (if the formal is preceded by the keyword `var`).
- In C, all parameters are passed by value. Pass by reference can be simulated by explicitly passing the address of a variable: `swap(&x, &y)`.
- In FORTRAN, all parameters are passed by reference. A programmer can simulate pass-by-value by explicitly making a local copy of an argument.
- Unlike most languages, FORTRAN allows r-values to be passed by reference: `swap(3+4, 7*x)`. The compiler creates a temporary variable to hold the value.

### Setting up Access Links...

**Case (1): Callee Within Caller:**

```plaintext
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 = 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.

**Case (2): Caller Within Callee:**

```plaintext
PROG M;
PROC Q(); ← N_Q = 1
PROC R();
PROC P(); ← N_P = 3
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.

### Parameter Passing...

- In Java, object references are transferred using pass-by-sharing. This means that the actual and formal will refer to the same object. The compiler simply passes the address of the object.
- In Java, primitive types are passed by value.

### Parameter Passing...

- In Java, object references are transferred using pass-by-sharing. This means that the actual and formal will refer to the same object. The compiler simply passes the address of the object.
- In Java, primitive types are passed by value.
Parameter Passing in Ada

Ada has three modes:
1. **in**-parameters pass information from the caller to the callee. The callee cannot write to them.
2. **out**-parameters pass information to the callee from the caller. The callee can read and write them. They start out being uninitialized.
3. **in out**-parameters pass information from the caller to the callee and back.

Parameter Passing in Ada...

For scalars and pointers, all modes should be implemented by copying values. Thus
1. **in**-parameters are passed-by-value.
2. **out**-parameters are passed-by-result (the formal is copied into the actual when the procedure returns).
3. **in out**-parameters are passed-by-value/result (On entry, the actual is copied into the formal. On return, the formal is copied back into the actual).

Parameter Passing...

In Pascal and Modula-2 a programmer would use **call-by-value** to
- ensure that the callee **cannot** modify the actual argument.

In Pascal and Modula-2 a programmer would use **call-by-reference** to
- ensure that the callee **can** modify the actual argument, or to
- make sure that a large parameter (which semantically should be passed by value) is not copied. (This is done for efficiency reasons).

Parameter Passing...

Modula-3 provides a **READONLY** parameter mode. A READONLY formal parameter cannot be changed by the callee. The formal
1. cannot be on the left-hand-side of an assignment statement, and
2. cannot be passed by reference to another routine.

- Small READONLY parameters are passed by value.
- Large READONLY parameters are passed by reference.
Parameter Passing in Ada...

For constructed types (records, arrays) an implementation is allowed to pass either values or addresses.

- If an in out parameter is passed by address an assignment to the formal changes the actual immediately.
- If an in out parameter is passed by value an assignment to the formal will not affect the actual until the procedure returns (and the formal is copied back into the actual).
- Ada disallows programs that can tell which implementation a compiler uses.

```ada
type t is record a, b : integer; end record;
r : t;

procedure foo (s : in out t) is
begin
    r.a := r.a + 1;
    s.a := s.a + 1;
end foo;

r.a := 3;
foo(r);
if r.a = 4 then put("implementation uses pass-by-value")
    else put("implementation uses pass-by-address")
```

Exam Problem 415.330/96 (A)

- Show the status of the run-time stack when execution has reached point for the second time in the program on the next slide.
- Fill in the name of each procedure invocation in the correct activation record. Also fill in the values of local variables and actual parameters, and show where the static links and control links are pointing.
- Assume that all actual parameters are passed on the stack rather than in registers.
PROGRAM M;
PROCEDURE P(X:INTEGER);
VAR A : INTEGER;
PROCEDURE Q(Y : INTEGER);
VAR B : INTEGER;
BEGIN
B := Y + 1; A := B + 2;
END Q;
BEGIN
P(B);
END Q;
BEGIN
A := X + 1; Q(A);
END P;
BEGIN
P(15); END M.

Homework

Draw the stack when control reaches point $\Diamond$ for the third time. Include all actual parameters, local variables, return addresses, and static and dynamic links.

Read Scott, pp. 441–448, 450–464
Read the Dragon Book: Parameter Passing 424–427
A parameter is often passed by the caller copying it (or its address, in case of VAR parameters) into the callee activation record. On the MIPS, the caller has an area in its own activation record in which it puts actual parameters before it jumps to the callee. For each procedure $P$ the compiler figures out the maximum number of arguments $P$ passes to any procedure it calls. The corresponding amount of memory has to be allocated in $P$’s activation record.