CSc 520

Principles of Programming Languages

28 : Control Structures — Parameters

Christian Collberg

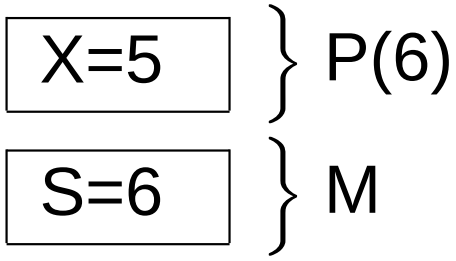
collberg+520@gmail.com

Department of Computer Science

University of Arizona

Parameter Passing – Value Parameters

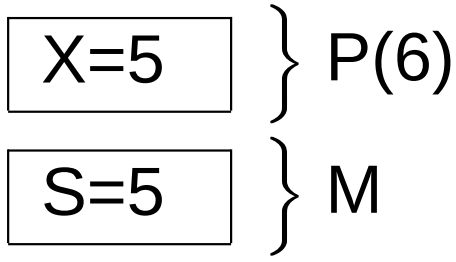
```
PROG M;
   PROC P(X:INT);
   BEGIN
      X:=5
   END P;
VAR S:INT;
BEGIN
   S:=6;
   P(S);
END.
```



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

Parameter Passing – Reference Parameter

```
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) fo 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-value*s 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
```


[4]

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-value*s in the *callee*'s activation record.

- The caller's actuals may be affected by the call.

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

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

Call-by-Name Parameters

- (Un-)popularized by Algol 60.
- A name parameter is (re-)evaluated
  - every time it is referenced,
  - in the callers environment.

Algorithm:

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.

Call-by-Name Parameters. . .

Algorithm:

4. If the parameter is used as an l-value, the thunk should return an l-value, otherwise an r-value.
5. 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.

Call-by-Name Parameters...

```
VAR i :  INTEGER; VAR a :  ARRAY 2 OF INTEGER;

PROCEDURE P (NAME x:INTEGER);
BEGIN
   i := i + 1; x := x + 1;
END;

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`.
- Since `i` is incremented before x, we get `a[2] := a[2] + 1`. $\Rightarrow$ Print `1,3`.

Call-by-Name – Implementation

```
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
```


[9]

Call-by-Name – Jensen's Device

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

VAR i :  INTEGER;
BEGIN
   WRITE Sum(i, i, 5);       (* ∑_{i=1}^{5} i  *)
   WRITE Sum(i*i, i, 10);    (* ∑_{i=1}^{10} i^2  *)
END
```

Large Value Parameters

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

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

Large Value Parameters. . .

Algorithm 1: Callee Copy

```
ROCEDURE P (VAR x :  T);
AR xT : T;
EGIN
  copy(xT,x,1000);
  xT[5]:="f";
ND P;
AR L : T;
EGIN
  P(L);
ND
```

Algorithm 2: Caller Copy

```
PROCEDURE P (VAR x :  T);
BEGIN
   x[5] := "f";
END P;
VAR L : T;
VAR LT : T;
BEGIN
   copy(LT, L, 1000);
   P(LT);
END
```

Parameter Passing

- In Pascal, parameters are passed either by value or by reference (if the formal is preceeded 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.

Parameter Passing. . .

- In Java, object references are transfered 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 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

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 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.

Parameter Passing in Ada. . .

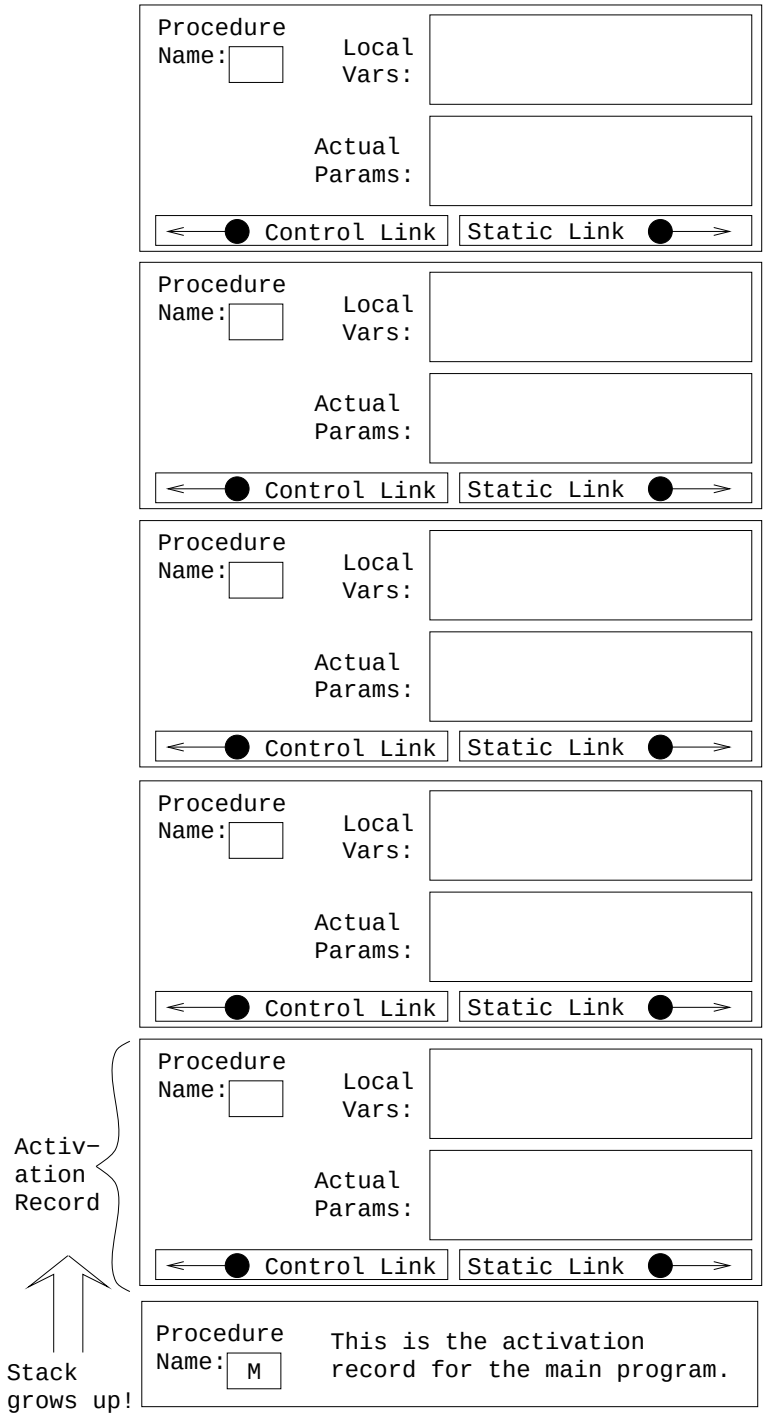
```
type t is record a, b :  integer; end record;
r :  t;

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.

Exam Problem 415.330/96 (B)

```
PROGRAM M;
 PROC P (
  X:INT);
  VAR W:INT;
  PROC Q (
   VAR Z:INT);
   VAR N:INT;
   PROC R (
    V:INT);
    VAR L:INT;
   BEGIN
    L := W;
    P(23);
   END R;
  BEGIN
   N := W;
   Z := Z+1;
   ◇
   R(Z);
  END Q;
 BEGIN
  W := X + 1;
  Q(W);
 END P;
BEGIN
 P(15);
END M.
```



Homework

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

```
PROGRAM M;
   PROCEDURE P(X:INTEGER);
   VAR A : INTEGER;
      PROCEDURE Q(Y : INTEGER);
      VAR B : INTEGER;
      BEGIN
         B := Y + 1; A := B + 2;
         ◇
         P(B);
      END Q;
   BEGIN
      A:= X + 1; Q(A);
   END P;
BEGIN P(0); END M.
```

Readings and References

- Read Scott, pp. 417–431

Summary

- A parameter is often passed by the caller copying it (or its address, in case of **VAR** parameters) into the callees 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.