

CSC 324: Principles of Programming Languages

Procedural Language Design Issues

©Suzanne Stevenson 2001
Modified by Sheila McIlraith 2004

Procedural Language Design Issues

Procedures: A Control Abstraction

- A block of code that can be called (imperative)
- A lambda expression (functional)
- A horn clause (logic programming)

Procedures modularize program structure

1

2

Components of a Procedure

1. Name
2. Formal parameters, optionally with types
 - **parameter** (formal parameter)
Local variable whose value is received from caller
 - **argument** (actual parameter)
The info passed from caller to callee
3. Body, which is a syntactic construct in the language:
 - Block, i.e., declarations and statements
 - Expression
 - Conjunction of terms
4. Optional result, optionally with a type

Procedure Implementation Issues

The general notion of a procedure leaves a number of points unspecified:

- How to pass parameters when the procedure is called
- How to maintain local state and control information
- How to access non-local names within a procedure body

3

4

Parameter Passing

Matching arguments with parameters:

1. Positional association:

- Arguments are associated with parameters left to right

2. Keyword association:

- Arguments are given tags, eg:
procedure plot (x,y: real; penup: boolean)
...
plot(0.0, 0.0, penup=>true)
plot(penup=>true, x=>0.0, y=>0.0)

5

Parameter Passing

3. Optional arguments:

- E.g., C printf(...)
- Extra arguments are packaged into some structure
- Passed to special parameter

6

Passing Modes

How to treat arguments

(pass-by-x/call-by-x):

1. Pass by value
(Java, C, C++, Pascal, Ada, Scheme, Algol68)
2. Pass by result
(Ada)
3. Pass by value-result
(some Fortrans, Ada)
4. Pass by reference
(Java objects, C++ with &, some Fortrans, Pascal with var, COBOL)
5. Pass by name
(Algol 60)

7

Example for Passing Modes

```
{ c : array[1..10] of integer;  
  m,n integer;  
  procedure r ( i , j : integer ) begin  
    i := i + 1;  
    j := j + 2  
  end r;  
  ...  
  m := 2;  
  n := 3;  
  r(m,n);           // call 1  
  write m, n ;     // print 1  
  
  m := 2;  
  c[1] := 1;  
  c[2] := 4;  
  c[3] := 8;  
  r(m,c[m]);       // call 2  
  write m,c[1],c[2],c[3]; // print 2  
}
```

8

Pass by Value

- Initial values of parameters copied from current values of arguments
- Final values of parameters are "lost" at return time (like local variables).
- Example:
at call 1: i = 2 j = 3
print 1:
at call 2: i = 2 j = 4
print 2:
- Benefit: Arguments protected from changes in procedure.
- Problem: Requires copying of values: costs time and space, especially for large aggregates.

9

Pass by Result

- No initial values of parameters
- Final values of parameters are copied back to arguments
- Example: does not work, as written

⇒ For **output** values only. Used to indicate that a parameter is intended solely for returning a result.

10

Pass by Result (Example)

Suppose proc r initializes i and j to 0:

- call 1:
 - final values of i and j:
 - m and n are set to:
- print 1:
- call 2: more problematic
 - final values of i and j:
 - which element of c is modified, c[1] or c[2]?
- print 2:
 - If c[1] is modified:
 - If c[2] is modified:

11

Problems with Pass by Result

- Requires copying of values: costs time and space, especially for large aggregates. (Cf. Call by value.)
- What if the argument is not a variable? E.g., r(1, 2);
- What if a variable is used twice in the argument list? E.g., r(m, m);
- What about calculations to determine locations of arguments? E.g., which c[m]?

12

Pass by Value-Result

- Initial values of parameters copied from current values of arguments
- Final values of parameters copied back to arguments

⇒ Combines functionality of pass by value and pass by result for **same** parameter.

13

Pass by Value-Result (Example)

- call 1:
 - initial: i = j =
 - final: i = j =
 - return: m and n set to:
- print 1:
- call 2:
 - initial: i = j =
 - final: i = j =
 - return: which element of c is modified, c[2] or c[3]?
- print 2:
 - if c[2] is modified:
 - if c[3] is modified:

14

Further Specifying Pass by Result

With pass by result or pass by value-result, order of assignments and address computations is important.

Options:

1. Perform return address computations at call time:
On second return:
m set to 3; c[2] set to 6
print 2:

15

Further Specifying Pass by Result (cont'd)

2. Perform return address computations at return time:

(a) Before any assignments:

On second return: same as above, but might not be if procedure has side-effects

(b) Just before that assignment, in order:

On second return:

m set to 3; c[3] set to 6

print 2:

16

Pass by Reference

- Formal parameters are pointers to the actual parameters (arguments).
- Address computations are performed at procedure call.
- Changes to the formal parameters are thus changes to the actual parameters.

17

Pass by Reference (Example)

- call 1:
 - initial: `i = j =`
 - final: `i = j =`
 - return: `m, n` are:
- print 1:
- call 2:
 - initial: `i = j =`
 - final: `i = j =`
 - return: `m, c[2]` are:
- print 2:

18

Pass by Reference

- Benefit: No copying for variables
- Problem: allow redefinition of expressions and constants?
- Problem: Leads to **aliasing**
 - two or more visible names for same location
 - can cause side effects not visible from code itself

19

Aliasing

```
{ y : integer ;
  procedure p ( x : integer ) begin
    x := x + 1;
    x := x + y
  end p;
  ...
  y := 2;
  p(y);
  write y
}
```

20

Aliasing

Pass by Reference:

- The identifiers `x` and `y` refer to the same location in call of `p`.
- Result of "write `y`"?

Pass by Value-Result:

- The identifiers `x` and `y` refer to different locations in call of `p`.
- Result of "write `y`"?

21

More Aliasing

```
{ i, j, k : integer ;
  procedure q ( a, b : integer ) begin
    a := i * b;
    b := i * b;
  end q;
  ...
  i := 2; j := 3; k := 4;
  q(i,j);
  q(k,k);
}
```

- First call has global-formal aliases:
 - `a` and `i`
- Second call has formal-formal alias:
 - `a` and `b`

22

Pass by Name

- A "name" for the argument is passed in to procedure
- Like textual substitution of argument in procedure
- Thus address computations are done whenever parameter is used
- Like pass-by-reference for scalar parameters

23

Pass by Name (Example)

- Example:
 - call 1: `m, n` set to:
 - print 1:
 - call 2: `m, c[m]` set to:
 - print 2:
- Benefit: same as pass by reference
- Problems: Inefficient, requires a *thunk*:
 - essentially a little program is passed that represents the argument
 - evaluates argument in caller's environment

24

Summary of Parameter Passing Modes

- Pass by value
- Pass by result
- Pass by value-result
- Pass by reference
- Pass by name