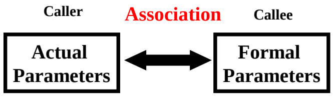
Subprograms

Fundamental Characteristics of Subprograms

- A subprogram has a single entry point.
- The caller is suspended during execution of the called subprogram.
- Control always returns to the caller when the called subprogram's execution terminates.

Parameters

- A **formal parameter**
  - A dummy variable listed in the subprogram header and used in the subprogram.
- An **actual parameter**
  - A value or address used in the subprogram call statement.



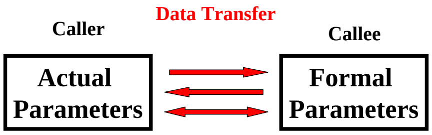
Subprograms: Procedures and Functions

- Two kinds of subprograms:
  - Procedures & Functions
- **Procedures** provide user-defined statements.
  - Abstraction over **statements**
- **Functions** provide user-defined operators.
  - Abstraction over **expressions**
- Most imperative languages provide both.

Subprograms – Design Issues

- What **p**arameter **p**assing **m**ethods are provided?
- Are parameter types checked?
- Are local variables static or dynamic?
- What is the referencing environment of a passed subprogram?
- Are parameter types in a passed subprogram checked?

Parameter Passing Methods



PPM: Semantic Models

- **In mode**
  - – FPs can receive data from the corresponding APs.
- **Out mode**
  - – FPs can transmit data to the corresponding APs.
- **In-out mode**
  - – FPs can receive/transmit data from/to the corresponding APs.

PPM: Transfer Model

- What transfer?
  - – An actual **value** is physically moved (transmitted).
  - – An access **path** to the value is moved (transmitted).
- When transfer:
  - – At the **entry**
  - – At the **exit**

Parameter Passing Methods

1. **Pass-by-Value**
2. **Pass-by-Result**
3. **Pass-by-Value/Result**
4. **Pass-by-Reference**
5. **Pass-by-Name**
6. **Pass-by-Text**

1. Pass-By-Value

- Copy-**in** the AP on **entry**.
  - – **in mode**
  - – Either by physical move or access path
  - – Disadvantages of access path method:
    - Must write-protect in the called subprogram
    - Accesses cost more (indirect addressing)
  - – Disadvantages of physical move:
    - Requires more storage
    - Cost of the moves

2. Pass-By-Result

- Copy-**out** the FP on **exit**.
  - – **out mode**
  - – Local's value is passed back to the caller
  - – Physical move is usually used
  - – Disadvantages:
    - If value is passed, time and space
    - In both cases, **order dependence** may be a problem

Pass-By-Result

- Problem:

```
sub(x, y)
…



sub(p1, p1)
```

3. Pass-By-Value/Result

- Copy-**in** the AP on **entry** & Copy-**out** the FP on **exit**
  - **inout mode**
  - Physical move, both ways
  - Disadvantages:
    - Those of pass-by-result
    - Those of pass-by-value

4. Pass-By-Reference

- Bind the reference of AP directly to FP.
  - **inout mode**
  - Pass an access path
  - Also called pass-by-sharing
  - Advantage:
    - Passing process is efficient
  - Disadvantages:
    - Slower accesses
    - Can allow aliasing

Pass-By-Reference

- Problem:
  - Aliasing
  - The called subprogram is provided wider access to nonlocals than is necessary.

Pass-By-Reference vs Pass-By-Value/Result

- **Pass-by-value-result does not allow these aliases (but has other problems!)**

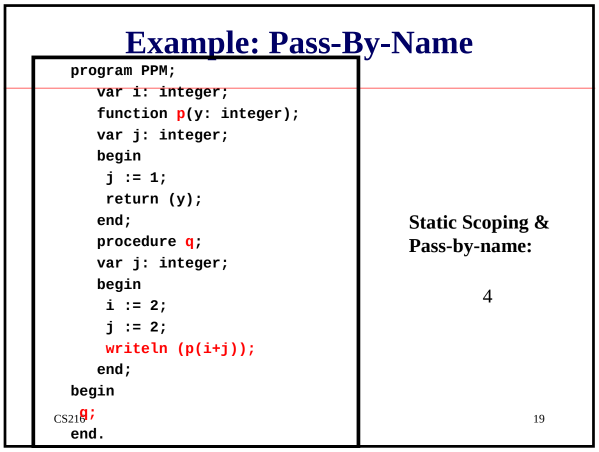
5. Pass-By-Name

- Substitute the expression of AP to FP.
  - **Inout mode**
  - By textual substitution of AP with FP
  - The AP is not evaluated until its use in the subprogram.
  - The AP will be evaluated **in the environment of the caller**.

Pass-By-Name

```
procedure p(x);
begin
  x := x + 1;
end;
…


p(a[i])
```

```
a[i] := a[i] + 1;
```

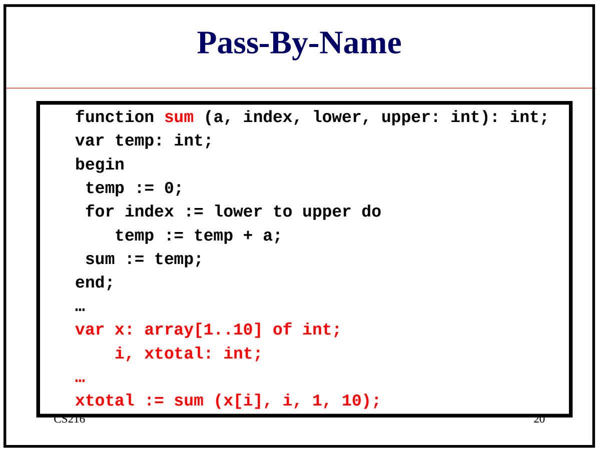


Example: Pass-By-Name

```
program PPM;
   var i: integer;
   function p(y: integer);
   var j: integer;
   begin
    j := 1;
    return (y);
   end;
   procedure q;
   var j: integer;
   begin
    i := 2;
    j := 2;
    writeln (p(i+j));
   end;
begin
   q;
end.
```

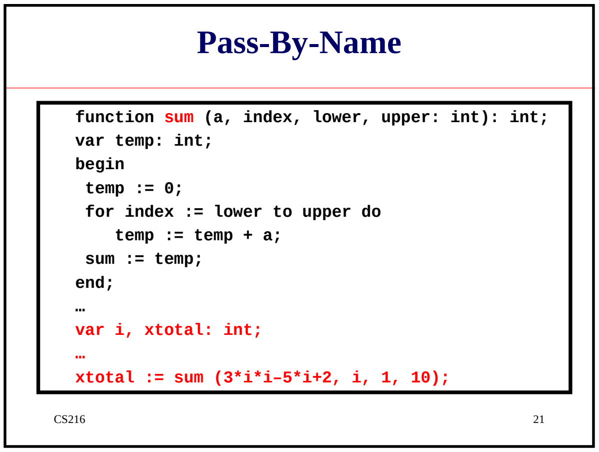
Static Scoping &
Pass-by-name:

4



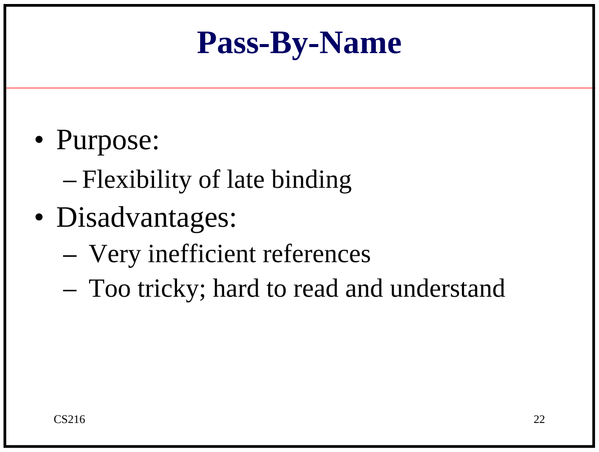
Pass-By-Name

```
function sum (a, index, lower, upper: int): int;
var temp: int;
begin
 temp := 0;
 for index := lower to upper do
    temp := temp + a;
 sum := temp;
end;
…
var x: array[1..10] of int;
    i, xtotal: int;
…
xtotal := sum (x[i], i, 1, 10);
```



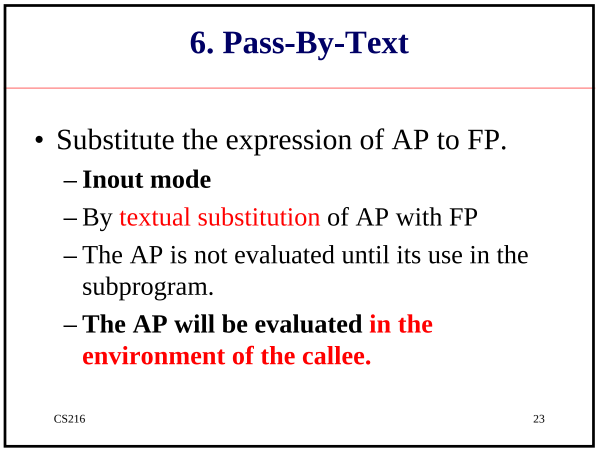
Pass-By-Name

```
function sum (a, index, lower, upper: int): int;
var temp: int;
begin
 temp := 0;
 for index := lower to upper do
    temp := temp + a;
 sum := temp;
end;
…
var i, xtotal: int;
…
xtotal := sum (3*i*i-5*i+2, i, 1, 10);
```



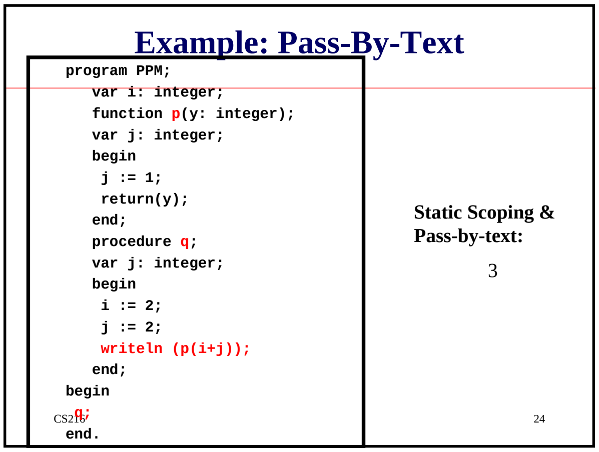
Pass-By-Name

- Purpose:
  - Flexibility of late binding
- Disadvantages:
  - Very inefficient references
  - Too tricky; hard to read and understand



6. Pass-By-Text

- Substitute the expression of AP to FP.
  - **Inout mode**
  - By textual substitution of AP with FP
  - The AP is not evaluated until its use in the subprogram.
  - **The AP will be evaluated in the environment of the callee.**



Example: Pass-By-Text

```
program PPM;
   var i: integer;
   function p(y: integer);
   var j: integer;
   begin
    j := 1;
    return(y);
   end;
   procedure q;
   var j: integer;
   begin
    i := 2;
    j := 2;
    writeln (p(i+j));
   end;
begin
   q;
end.
```

Static Scoping &
Pass-by-text:

3

Example 1: PPM under Static Scoping

```
program PPM;
   var i: integer;
       a: array[1..2] of integer;
   …
   procedure f ( x: integer);
      begin
         a[1] := 6;
         i := 2;
         x := x + 3;
end;
begin
   a[1] := 1;
   a[2] := 2;
   i := 1;
   f(a[i]);
   write( a[1], a[2], i);
end.
```

Pass-by-value:

6 2 2

Pass-by-result:

error

Example 1: PPM under Static Scoping

```
program PPM;
   var i: integer;
       a: array[1..2] of integer;
   …
   procedure f ( x: integer);
      begin
         a[1] := 6;
         i := 2;
         x := x + 3;
end;
begin
   a[1] := 1;
   a[2] := 2;
   i := 1;
   f(a[i]);
   write( a[1], a[2], i);
end.
```

Pass-by-value/result:

4 2 2

Pass-by-reference:

9 2 2

Example 1: PPM under Static Scoping

```
program PPM;
   var i: integer;
       a: array[1..2] of integer;
   …
   procedure f ( x: integer);
      begin
         a[1] := 6;
         i := 2;
         x := x + 3;
end;
begin
   a[1] := 1;
   a[2] := 2;
   i := 1;
   f(a[i]);
   write( a[1], a[2], i);
end.
```

Pass-by-name:

6 5 2

Pass-by-text:

6 5 2

Example 2: PPM under Static Scoping

```
program PPM;
   var i: integer;
       a: array[1..2] of integer;
   …
   procedure f ( x: integer);
      begin
         a[1] := 6;
         i := 2;
         x := x + 3;
end;
begin
   a[1] := 1;
   a[2] := 2;
   i := 1;
   f(a[i]);
   write( a[1], a[2], i);
end.
```

4 2 2 What PPM?

Pass-by-value/result

Example 2: PPM under Static Scoping

```
program PPM;
   var i: integer;
       a: array[1..2] of integer;
   …
   procedure f ( x: integer);
      begin
         a[1] := 6;
         i := 2;
         x := x + 3;
end;
begin
   a[1] := 1;
   a[2] := 2;
   i := 1;
   f(a[i]);
   write( a[1], a[2], i);
end.
```

6 5 2 What PPM?

Pass-by-name

Example 3: PPM under Static Scoping

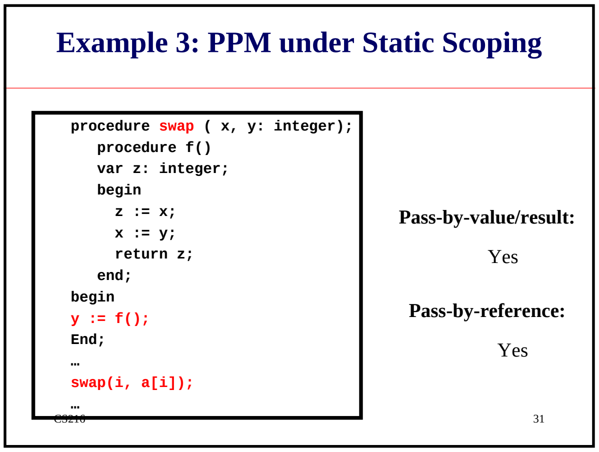
```
procedure swap ( x, y: integer);
   procedure f()
   var z: integer;
   begin
     z := x;
     x := y;
     return z;
   end;
begin
y := f();
End;
…
swap(i, a[i]);
…
```

Pass-by-value:

No

Pass-by-result:

No



Example 3: PPM under Static Scoping

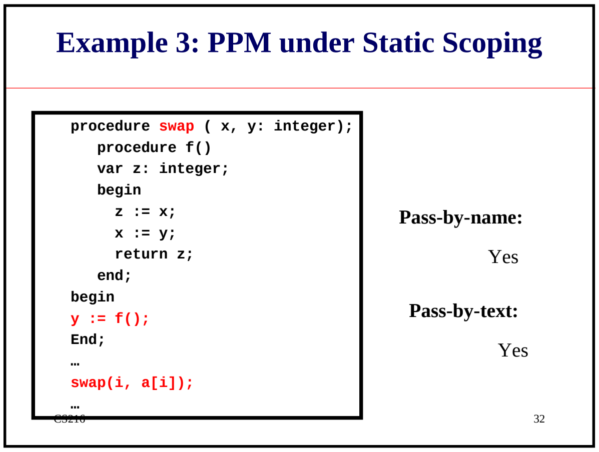
```
procedure swap ( x, y: integer);
   procedure f()
   var z: integer;
   begin
     z := x;
     x := y;
     return z;
   end;
begin
y := f();
End;
…
swap(i, a[i]);
…
```

Pass-by-value/result:

Yes

Pass-by-reference:

Yes



Example 3: PPM under Static Scoping

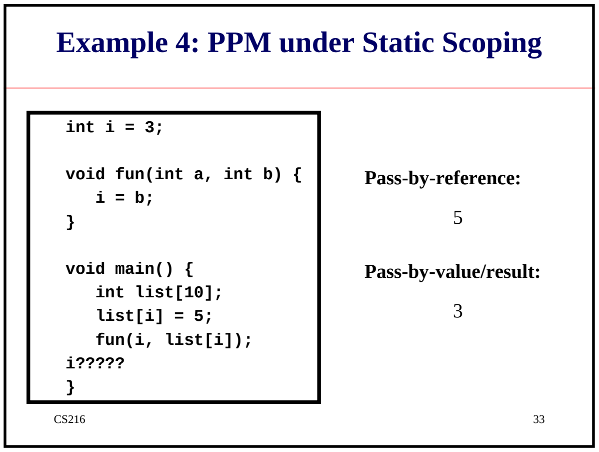
```
procedure swap ( x, y: integer);
   procedure f()
   var z: integer;
   begin
     z := x;
     x := y;
     return z;
   end;
begin
y := f();
End;
…
swap(i, a[i]);
…
```

Pass-by-name:

Yes

Pass-by-text:

Yes



Example 4: PPM under Static Scoping

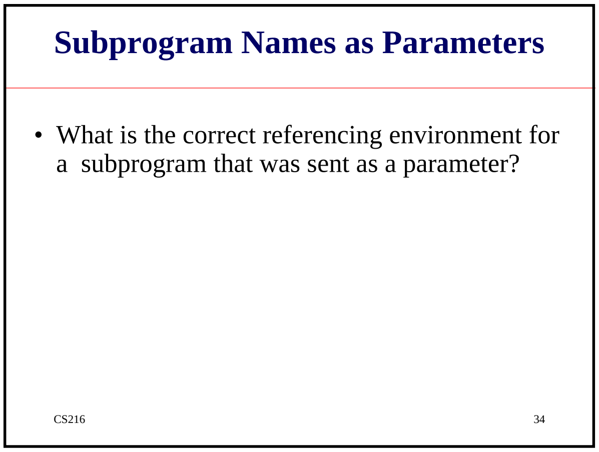
```
int i = 3;

void fun(int a, int b) {
   i = b;
}

void main() {
   int list[10];
   list[i] = 5;
   fun(i, list[i]);
i?????
}
```

Pass-by-reference:

5

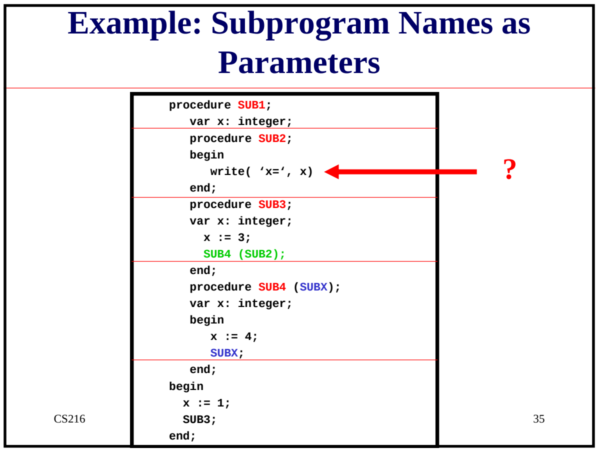
Pass-by-value/result:

3



Subprogram Names as Parameters

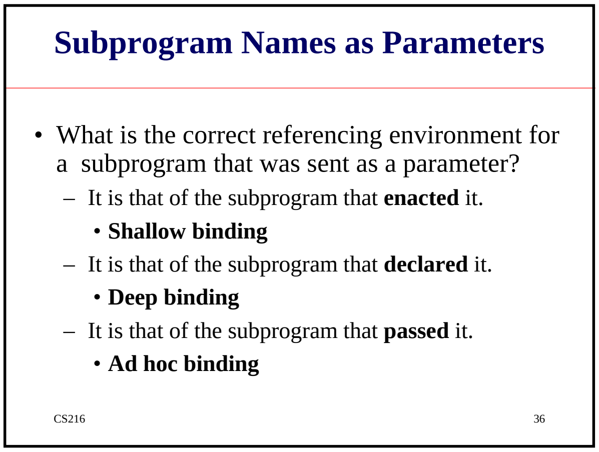
- What is the correct referencing environment for a subprogram that was sent as a parameter?



Example: Subprogram Names as Parameters

```
procedure SUB1;
   var x: integer;
   procedure SUB2;
   begin
      write( ‘x=‘, x)
   end;
   procedure SUB3;
   var x: integer;
     x := 3;
     SUB4 (SUB2);
   end;
   procedure SUB4 (SUBX);
   var x: integer;
   begin
      x := 4;
      SUBX;
   end;
begin
  x := 1;
  SUB3;
end;
```

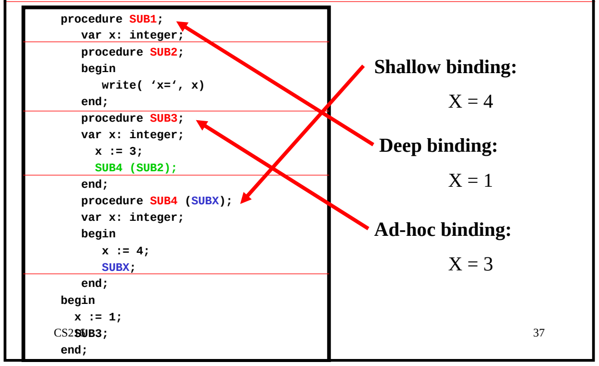
?



Subprogram Names as Parameters

- What is the correct referencing environment for a subprogram that was sent as a parameter?
  - It is that of the subprogram that **enacted** it.
    - **Shallow binding**
  - It is that of the subprogram that **declared** it.
    - **Deep binding**
  - It is that of the subprogram that **passed** it.
    - **Ad hoc binding**

Example: Subprogram Names as Parameters



```
procedure SUB1;
   var x: integer;
   procedure SUB2;
   begin
      write( ‘x=‘, x)
   end;
   procedure SUB3;
   var x: integer;
     x := 3;
     SUB4 (SUB2);
   end;
   procedure SUB4 (SUBX);
   var x: integer;
   begin
      x := 4;
      SUBX;
   end;
begin
  x := 1;
  SUB3;
end;
```

Shallow binding:

X = 4

Deep binding:

X = 1

Ad-hoc binding:

X = 3

Subprogram Names as Parameters

- For static-scoped languages,
  - – Deep binding is most natural.
- For dynamic-scoped languages,
  - – Shallow binding is most natural.