Parameter passing, scope
Parameter passing mechanisms

1. pass-by-value
2. pass-by-reference
3. pass-by-value-result
4. pass-by-name / macro expansion

• See Sethi 5.2-5.3
Formal and actual parameters

procedure p(x, y : int) {
    x := 7;
    y := x + 1;
}

a := 2;
p(a, a+1);

formal parameters

actual parameters
Call-by-value

procedure p(x, y : int) {
    x := 7;
    y := x + 1;
}

a := 2;
p(a, a+1);

values of actuals get copied into formals at time of call
Call-by-reference

```plaintext
procedure p(x, y : int) {
    x := 7;
    y := x + 1;
}

a := 2;
p(a, a+1);
```
Call-by-reference

```plaintext
procedure p(x, y : int) {
    x := 7;
    y := x + 1;
}

a := 2;  b := 5;
p(a, b);

locations of actuals get passed at time of call
```
Call-by-value-result

• aka “copy-in / copy-out”
• used in Ada
• same as call-by-reference, except actuals are not modified until the end
Call-by-value-result

procedure p(x, y : int) {
    x := 7;
    y := x + 1;
}

a := 2;  b := 5;
p(a, b);

*values* of actuals get *copied in* at time of call, and *copied out* at end
Call-by-value-result

• aka “copy-in / copy-out”
• used in Ada
• same as call-by-reference, except actuals are not modified until the end

• can produce different results from call-by-reference when have aliases (two names for the same location)
  • (considered erroneous in Ada)
Trouble: aliases

```plaintext
var i;
proc p(x) {
    i := 0;
}
i := 2;
p(i);
print(i);
```

- x and i are aliases
- call-by-reference:
  prints 0
- call-by-value-result:
  prints 2
  (since x=2 is written back at end)
Call-by-name

• used by Algol 60
• textual substitution:
  • paste procedure body into program with parameters renamed

```
proc swap(a, b) {
  var t;
  t := a; a := b; b := t;
}

... swap(x, y); ...
```

```
var t;
t := x; x := y; y := t;
...```

Call-by-name

• used by Algol 60
• textual substitution:
  • paste procedure body into program with parameters renamed
• Algol had complicated rules for renaming variables
• Call-by-name not used anymore
• But, closely related:
Macro expansion

- C has preprocessor for textual substitution using \#define

\#define BUF_SIZE 1000

char buf[BUF_SIZE]; \textcolor{blue}{\rightarrow} \textcolor{green}{char buf[1000];}
Macro expansion

- C has preprocessor for textual substitution using `#define`

```c
#define SQR(x) (x * x)

int i = SQR(7);  // int i = (7 * 7);
```
Macro expansion

• C has preprocessor for textual substitution using `#define`
• careful!

```c
#define SQR(x) (x * x)

int i = SQR(a + b);  // int i = (a + b * a + b);
```
Macro expansion

• C has preprocessor for textual substitution using \#define
• better:

```c
#define SQR(x) ((x)*(x))

int i = SQR(a + b);  \rightarrow  int i = ((a + b)*(a + b));
```
[Break]
Scope  (Sethi Ch 5.3, 5.4)

• Scope of a variable = where it is “visible”
• var x : integer;  // declaration of x
• x := 7;       // use of x

• compiler binds each use to the corresponding declaration
• if multiple declarations, use closest “enclosing” one
Scope

• Scope of a variable = where it is “visible”

• var x : integer;     // declaration of x

• x := 7;                    // use of x

• compiler binds each use to the corresponding declaration

• if multiple declarations, use closest “enclosing” one

Figure 5.9 Each procedure declaration is enclosed in a box, as is the whole program.
Lexical vs dynamic scope

• Lexical scope: all bindings from uses to declarations known at compile time

• Dynamic scope: bindings occur at run time

• Macro expansion (textual substitution) can yield dynamic scope

```plaintext
int n = 7;
#define PRINT print(n)

proc p() {
    int n = 5;
    PRINT;
}

PRINT;
```
Parameter passing puzzle

• what is printed with
  1. call-by-value?
  2. call-by-reference?
  3. call-by-value-result?
  4. macro expansion?

```plaintext
proc swap(a, b) {
    var t;
    t := a; a := b; b := t;
}

i := 2;   a[2] := 10;
swap(i, a[i]);

print(i, a[2]);
```
Parameter passing puzzle

• what is printed with
  1. call-by-value?  
     \[
     \begin{align*}
     &2 \quad 10
     \end{align*}
     \]
  2. call-by-reference?  
     \[
     \begin{align*}
     &10 \quad 2
     \end{align*}
     \]
  3. call-by-value-result?  
     \[
     \begin{align*}
     &10 \quad 2
     \end{align*}
     \]
  4. macro expansion?  
     \[
     \begin{align*}
     &10 \quad 10
     \end{align*}
     \]

```plaintext
proc swap(a, b) {
    var t;
    t := a; a := b; b := t;
}
i := 2;  a[2] := 10;
swap(i, a[i]);
print(i, a[2]);
```