Parameter Passing Methods

References:

- Sebesta, sections 8.1–8.5
- Sethi, sections 5.1–5.2

Pass-by-value

Method: Evaluate the actual parameter and pass its value to the formal parameter. Changes to it may be allowed inside the subprogram, but they do not affect the actual parameter.

Implementations:

- copy the value from actual to formal parameter.
- send a reference to the formal parameter, but write-protect what it refers to.

Advantages:

Disadvantages:

Pass-by-reference

Method: Compute a reference to the actual parameter and pass that to the formal parameter. Any change inside the subprogram to the thing referred to stays in effect when you return to the caller.

(Notice that we're mixing up abstract specifications and implementation details. Ideally, the specifications for parameter passing would not mention implementation details.)

Pass-by-reference can't be used to pass a literal value. Why not?

What's going on here?

```
// C code:
public inc( int* i ) {
    (*i)++;
}

public main () {
    int n = 50;
    inc( &n );
}
```

```
// Java code:
class Fraction {
   private int num, denom;
   public static void multiply
         ( Fraction f, int n ) {
       f.num *= n;
   }
}
Fraction g = new Fraction( 5,7 );
Fraction.multiply(g, 2);
```

Advantages:

Disadvantages:

Pass-by-value-return

Method: Evaluate the actual parameter and pass its value to the formal parameter. Changes to it are allowed inside the subprogram. Upon return, evaluate the formal parameter and pass its value back to the actual parameter.

Advantages:

Disadvantages:

Pass-by-value-result vs Pass-by-reference

They don't *always* return the same answer!

```
int i;

subprogram foo( int x, int y ){
    i = y;
}

main() {
    i = 2;
    j = 3;
    foo( i, j );
}
```

```
class X {
   private int t;

   subprogram set(int n) {
       t = n;
   }

   int subprogram square (int a) {
       set(3);
       ans = a * a;
       a = 1;
       return ans;
   }

   main() {
       t = 11;
       print square(t);
   }
}
```


Pass-by-name

Method:

- Textually substitute the actual parameter into every occurrence of the formal parameter in the subprogram.
- Rename locals, if necessary, to avoid name conflicts.
- During subprogram execution, evaluate the parameter (whenever encountered) in the context of the calling environment.

Implementation: Use a call to a parameterless subprogram (called a "thunk") where the formal parameter occurs.

Example:

```
subprogram swap (int a, int b) {
    int temp;
    temp = a;
    a = b;
    b = temp;
}

main() {
    int p = 12;
    int q = -4;
    swap(p, q);
    // Create list, an array of 5, containing: 1,3,5,7,9
    p = 888;
    swap(p, list[0]);
    p = 3;
    swap(p, list[p]);
}
```

Advantages:

Disadvantages: