Parameter Passing Methods

Procedural abstraction
• Parameter passing methods
  – pass by value
  – pass by result
  – pass by value-result
  – pass by reference
    • aliasing
  – pass by name
• Procedures/functions as arguments

Procedures
• Modularize program structure
  – Argument: information passed from caller to callee (actual parameter)
  – Parameter: local variable whose value (sometimes) is received from caller (formal parameter)
• Procedure declaration
  – name, formal parameters, procedure body with local declarations and statement list, optional result type

```c
void translateX(point *p, int dx)
```
Parameter Association

- **Positional association**
  - Arguments associated with formals one-by-one
    - E.g., C, Pascal, Scheme, Java
- **Keyword association**
  - E.g., Ada uses a mixture
    ```
    procedure plot (x,y: in real; penup: in boolean)
    .... plot (0.0, 0.0, penup=> true)
    ....plot (penup=>true, x=>0.0, y=>0.0)
    ```

Parameter Passing Modes

- **pass by value**
  - C, Pascal, Ada, Scheme, Algol68
- **pass by result**
  - Ada
- **pass by value-result (copy-in, copy-out)**
  - Fortran, sometimes Ada
- **pass by reference**
  - Fortran, Pascal `var` params, sometimes Cobol
- **pass by name** (outmoded)
  - Algol60
Pass by Value

{ c: array [1..10] of integer;
m,n : integer;
procedure r (k,j : integer)
begin
  k := k+1;
  j := j+2;
end r;
...
m := 5;
n := 3;
r(m,n);
write m,n;
}

Advantages
– Argument protected from changes in callee

Disadvantages
– Copying of values takes execution time and space, especially for aggregate values
Pass by Result

{ c: array [1..10] of integer;
m,n : integer;
procedure r (k,j : integer)
begin
    k := k+1;
    j := j+2;
end r;
...

m := 5;
n := 3;
r(m,n);
write m,n;
}

Pass by Result

- Assume we have procedure \( p(k, j : \text{int}) \) with \( k \) and \( j \) as result parameters. What is the interpretation of \( p(m,m) \)?
- Assume parameter \( k \) has value 2 and \( j \) has value 3 at end of \( p \). What value is \( m \) on return?
Pass by Value-Result

```
{ c: array [1..10] of integer;
  m,n : integer;
  procedure r (k, j : integer) begin
    k := k+1;
    j := j+2;
  end r;

  m := 5;
  n := 3;
  r(m, n);
  write m,n;
}
```

Output:
6 5

Pass by Value-Result

```
{ c: array [1..10] of integer;
  m,n : integer;
  procedure r (k, j : integer) begin
    k := k+1;
    j := j+2;
  end r;

  m := 5;
  n := 3;
  r(m, c[m]);
  write c[1], c[2], ..., c[10];
}
```

What element of c has its value changed? c[2]? c[3]?
Pass by Reference

\[
\begin{array}{l}
\{ \ c: \text{array [1..10] of integer;} \\
m, n : \text{integer}; \\
\text{procedure r (k, j : integer)} \\
\text{begin} \\
\quad k := k + 1; \\
\quad j := j + 2; \\
\text{end r;} \\
\ldots \\
m := 5; \\
n := 3; \\
r(m, n); \\
\text{write m, n;} \\
\} \\
\end{array}
\]

Comparisons

- **Value-result**
  - Has all advantages and disadvantages of value and result together

- **Reference**
  - Advantage: is more efficient than copying
  - Disadvantage: can redefine constants
    
    \[ r(0, X) \text{ will redefine the constant zero in old Fortran’66 compilers} \]
  - Leads to aliasing: when there are two or more different names for the same storage location
    - Side effects not visible from code itself
Aliasing: by Reference

{ y: integer;
 procedure p(x: integer)
 { x := x + 1;
   x := x + y;
 }
...
 y := 2;
 p(y);
 write y;
} output: 6
during the call,
x and y are the same location!

No Aliasing: Value-Result

{ y: integer;
 procedure p(x: integer)
 { x := x + 1;
   x := x + y;
 }
...
 y := 2;
 p(y);
 write y;
} output: 5
Another Aliasing Example

\{  j, k, m :integer;
    procedure q( a, b: integer)
    \{  b := 3;
        m := m *a;
    \}
    ...
    \{  s1:  q(m, k);
        ...
        \{  s2: q(j, j);
            ...
            \}
    \}

<table>
<thead>
<tr>
<th>global-formal aliases:</th>
</tr>
</thead>
<tbody>
<tr>
<td>\langle m, a \rangle &lt; k, b &gt; associations during call S1;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>formal-formal aliases:</th>
</tr>
</thead>
<tbody>
<tr>
<td>\langle a, b \rangle during call S2;</td>
</tr>
</tbody>
</table>

Pass by Reference

- Disadvantage: if an error occurs, harder to trace values since some side-effected values are in environment of the caller
- What happens when someone uses an expression argument for a by reference parameter?
  - \((2*x)??\)
Pass by Name

```
{ c: array [1..10] of integer;
m,n : integer;
procedure r (k,j : integer)
begin
  k := k+1;
j := j+2;
end r;
/* set c[n] to n */
m := 2;
r(m,c[m]);
write m,n;
}
```

Pass by Name

- **Algol60 device**
  - Deferred calculation of the argument until needed; like textual substitution with name clashes resolved
  - **THUNK** - evaluates argument in caller’s environment and returns address of location containing the result

- **Characteristics**
  - Inefficient
  - Same as pass by reference for scalars
Procedures as Parameters

- To type check the call, need the full function signature of the function argument
  
  `<function name> : <vector of parameter types> →<return type>

  e.g., `translateX: (point *, int) → void

  procedure q( x: integer;
  function s (y,z: integer):integer)

  s takes 2 integer arguments and returns an integer!

Example

```plaintext
{ m, k : integer;
procedure q(x : integer; function s(y,z: integer): integer)
{  k, l : integer;
    ...
    s(…); /*call to function parameter s */
    ...
} /* end of q*/
integer function f(w,v: integer)
{  ...
    w := k*v; /* which k is this? k or k*/
}  
...
q(m, f);
... 
} 
```