Parameter Passing Methods

Iqbal Mohomed

Formal vs Actual Parameters

```c
void write(int f)
{
    // Blah
}

void main()
{
    int a = 3;
    write (a);
}
```

- We call a the actual parameter
- We call f the formal parameter

Semantic Models of Parameter Passing

- in-mode: FP can receive data from AP
- out-mode: FP can transmit data to AP
- inout-mode: Do both

Conceptual Model of Data Transfer

- Transmission of actual value (copy)
- Transmission of access path (alias)
Pass-by-Value

• Implements in-mode semantics
• Formal parameter is a new local variable that exists within the scope of the procedure/function/subprogram
• Value of actual parameter is used to initialize the formal parameter
• Changes made to formal parameter DO NOT get transmitted back to the caller through parameter passing!

Pass-by-Value (contd)

• Typically implemented by copy
  – Inefficiency in storage allocation
  – inefficiency in copying value
  – For objects and arrays, the copy semantics are costly!
• If access-path were transmitted ...
  – Need to write-protect cell, which is tricky

Pass-by-Result

• Implements out-mode semantics
• Formal parameter is a new local variable that exists within the scope of the function
• No value is transmitted to the function
• Just before control is transferred back to the caller, the value of the formal parameter is transmitted back to the actual parameter

Pass-by-Result (contd)

• Actual parameter MUST be a variable
• Parameter collisions can occur!
  – write(p1,p1); .. if the two formal parameters in write had different names, which value should go into p1?
  – Order in which actual parameters are copied determines their value!
• Typically implemented by copy
  – Inefficiency in storage allocation
  – inefficiency in copying value
  – For objects and arrays, the copy semantics are costly!
  – The added problem is that we don’t want to use the value of the actual parameter as the initial value of the formal parameter
Pass-by-Result (contd)

• When is the address of the actual parameter evaluated?

```java
write(list[index]);
```

• Suppose index is a global variable that is modified by write. Should we transmit the result to list[index₁] or list[index₂]?

Pass-by-Value-Result

• Implements inout-mode semantics
• Combination of Pass-by-Value and Pass-by-Result
• Formal parameter is a new local variable that exists within the scope of the function
• Value of actual parameter is used to initialize the formal parameter
• Just before control is transferred back to the caller, the value of the formal parameter is transmitted back to the actual parameter

Pass-by-Value-Result (contd)

• Shares with Pass-by-Value and Pass-by-Result the disadvantage of requiring multiple storage parameters and time for copying values
• Shares with Pass-by-Result the problem associated with the order in which actual parameters are assigned

Pass-by-Reference

• Implements inout-mode semantics
• Formal parameter is an alias for the actual parameter
• Changes made to formal parameter DO get (effectively) transmitted back to the caller through parameter passing!
<table>
<thead>
<tr>
<th>Pass-by-Reference (contd)</th>
<th>Pass-by-Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Efficient in both time and space!</td>
<td>• Implements inout-mode semantics</td>
</tr>
<tr>
<td>• Disadvantage is aliasing</td>
<td>• The actual parameter is textually substituted for the corresponding formal parameter in all its occurrences within the function</td>
</tr>
<tr>
<td>– Many potential scenarios can occur</td>
<td>• Based on the concept of late-binding</td>
</tr>
<tr>
<td>– Harmful to program readability and thus to reliability</td>
<td>– Lazy evaluation is also based on the idea of late-binding ... short-circuit evaluation is an example of lazy evaluation</td>
</tr>
</tbody>
</table>

### Parameter Passing in Major Languages

| C and Java both use pass-by-value semantics. |
| People keep saying objects are treated as pass-by-reference in Java. Technically ... that is WRONG! |
| C++ uses pass-by-value as the default convention but allows pass-by-reference parameters as well |
|   – the & operator is used to indicate pass-by-reference |
|   – e.g. void write(int& p){ ///} |