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

    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)
  • Similarly for common lisp
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)  By Value:
  begin
    k := k+1;
    j := j+2;
  end r;
...
  m := 5;
  n := 3;
  r(m,n);
  write m,n;
}

Output:

5    3
Pass by Value

• Advantages
  – Argument protected from changes in callee

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

\{
  \begin{align*}
    \text{c: array [1..10] of integer;}
    \text{m,n : integer;}
    \text{procedure r (k,j : integer)}
    \begin{align*}
      \text{begin}
      \quad k := k+1; & \text{Error in procedure r:} \\
      \quad j := j+2; & \text{can’t use parameters which} \\
      \text{end r; \quad are uninitialized!} & \text{are uninitialized!}
    \end{align*}
  \end{align*}
\}

... \begin{align*}
  \text{m := 5;}
  \text{n := 3;}
  \text{r(m,n);}
  \text{write m,n;}
\end{align*}
Pass by Result

• Assume we have \textit{procedure} \texttt{p(k, j : int)} with \texttt{k} and \texttt{j} as result parameters. what is the interpretation of \texttt{p(m,m)}?
  – Assume parameter \texttt{k} has value 2 and \texttt{j} has value 3 at end of \texttt{p}. What value is \texttt{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;
\}

By Value-Result

\[
\begin{array}{c|c}
  k & j \\ \hline
  5 & 3 \\
  6 & 5 \\
\end{array}
\]

Output:

Output:

\[
\begin{array}{c|c}
  m & n \\ \hline
  6 & 5 \\
\end{array}
\]
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;
    /* set c[m] = m */
    m := 2;
    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

\{
  \texttt{c: \text{array } [1..10] \text{ of integer;}}
  \texttt{m,n : \text{integer;}}
  \texttt{procedure r (k,j : \text{integer})}
  \texttt{begin}
    \begin{align*}
      k & := k+1; \\
      j & := j+2;
    \end{align*}
  \texttt{end r;}
\}

\...
\texttt{m := 5;}
\texttt{n := 3;}
\texttt{r(m,n);}
\texttt{write m,n;}

\texttt{Value update happens in storage of the caller while callee is executing}
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;
}

during the call, x and y are the same location!

output: 6
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);
... 
}
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;
}

m:= m+1
m  c[
2      1 2 3 4 5 6 7 8 9 10
3 1 2 5 4 5 6 7 8 9 10
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

  \(<function\ name>:\\n  <vector\ of\ parameter\ types> \ \square \ \langle return\ type\rangle\\n\)

  e.g., `translateX: (point *, int) \ \square \ \text{void}`

procedure `q( x: integer;`

  `function s (y,z: integer):integer)\)

  \textit{s takes 2 integer arguments and returns an integer!}`
Example

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