1 - Parameter Passing

- ...how a subroutine's parameter actually gets the argument value
- (odd? but the name is probably based on some alternate terminology:
  - another name for parameter: formal parameter
  - another name for argument: actual parameter)

2 - Parameter Passing in FORTRAN

- FORTRAN only provided a single parameter-passing mode;
- If I am reading MacLennan, Ch. 2, correctly, this can vary depending on the FORTRAN implementation (!)
- HOWEVER, usually the parameter is bound to the address of the argument;
  - so, if the parameter is changed, so is the argument;
- This parameter-passing mode should already be familiar to you ... it is more commonly known as...

3 - Pass-By-Reference

- So: consider this FORTRAN subroutine:

```
SUBROUTINE DIST (D, X, Y)
D = X - Y
IF (D .LT. 0) D = -D
RETURN
END
```

- ...and this fragment calling this subroutine:

```
DIST1 = 10
X1 = 20
Y1 = 30
CALL DIST (DIST1, X1, Y1)
```

- because parameter `D` is bound to argument `DIST1`'s address, when `D` is set to -10 and then 10, so is `DIST1`

4 - Pass-By-Reference - Advantages

- can be used for input *and* output *and* input/output parameters
- it is always reasonably efficient -- just copy over an address (remember, efficiency was an important goal for FORTRAN)
  - why just "reasonably"? Well, there is still a slight cost for indirection --
  - [MacLennan, p. 58] "Instead of the subprogram having the value of the actual parameter directly available, it only has the address of the actual"
  - "Therefore, an extra memory reference is required to fetch or store the value of a parameter passed by reference"

5 - Pass-By-Reference - Disadvantages

- ...it can have dangerous consequences
- one may ACCIDENTALLY change an argument one didn't MEAN to;
- in early FORTRAN, MacLennan says that you COULD even change a CONSTANT used as an argument this way...! (p. 59)

6 - Pass-By-Value-Result

- another way of implementing FORTRAN's parameter passing; arguably a bit more secure
- when the subroutine is called, the **value** of the argument is copied into the parameter;
- when the subroutine exits, the **result**, or value of the parameter at the time of the subroutine exit, is copied into the argument;
- yes, the argument is still changed -- BUT since both of these operations are done by the caller (accordingly to MacLennan, p. 60), the compiler can know more easily to omit the copying of the result into the argument if that argument is a constant or expression (that is, a value that it is not reasonable to change)

7 - Pass-By-Value-Result vs. Pass-By-Reference

- pass-by-value-result OFTEN gives the same result as pass-by-reference -- but NOT ALWAYS;
- ...if aliasing is involved, for example, you can get different results with these two modes;
- consider this pseudocode (from Scott, p. 395):

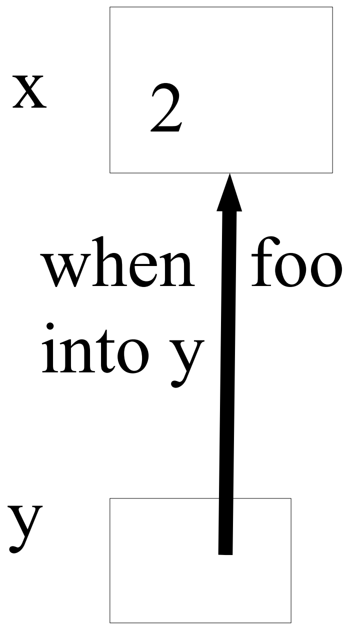
```
x: integer;

procedure foo (y: integer)
    y := 3
    print x

...
x := 2
foo(x)
print x
```

8 - comparative example p. 1

- if x is passed by reference?



- when foo(x) is called, x's address is copied into y
- see how, when y is changed within foo, x is immediately changed?
- so, printing x within foo prints 3, as does printing x after foo is complete;
- what this fragment outputs for pass-by-reference:

```
3
3
```

9 - comparative example p. 2

- if x is passed by value-result?

x $\boxed{2}$

- when foo(x) is called, x's value is copied into y

y $\boxed{2}$

- see how, when y is changed within foo, x is NOT immediately changed?
- so, printing x within foo prints 2
- BUT -- when foo is complete, y's value, 3, is copied back into argument x; NOW x becomes 3. Printing x after foo will show 3.
- what this fragment outputs for pass-by-value-result:

```
2
3
```

10 - Algol-60's approach

- [MacLennan p. 129] "...the problem with FORTRAN's parameters results from the failure to distinguish parameters intended for **input** from those intended for **output** (or **both** input and output)."
- "Algol-60 attempted to solve this problem by providing **two** parameter-passing modes:
  - **pass-by-value** for input parameters and
  - **pass-by-name** for all other kinds of parameters."
- **Pass-by-value** you should already know -- that's the default for C++ scalar parameters:
  - the value of the argument is COPIED to the parameter;
  - ...change the parameter? You only change that local copy, NOT the argument

11 - Algol-60's approach, cont'd

- interestingly, pass-by-name is the DEFAULT for Algol-60, and you only get pass-by-value if you explicitly SPECIFY it:

```
real procedure Sum (k, l, u, ak)
    value l, u;
    integer k, l, u;
    real ak;
    begin
        ...
    end;
```

- see how `l` and `u` are specified to be pass-by-value?
- since this isn't specified for `k` and `ak`, those are pass-by-name

12 - pass-by-name, p. 1

- consider the so-called copy rule for subroutine invocation:
  - a subroutine call can be replaced by its body with the parameters replaced by its arguments
- EXCEPT -- we know, from experience, that this is NOT always the case for pass-by-value or pass-by-reference or pass-by-value-result!
- ...pass-by-name attempts to REALLY do this, though!

13 - pass-by-name, p. 2

- Consider:

```
procedure Inc(n);
    integer n;
    begin
        n := n + 1;
    end;
```

- with pass-by-name, the call:

  ```
  Inc(i)
  ```

  ...should have the same effect as if it were replaced with:

  ```
  i := i + 1;
  ```

- ...and the call

  ```
  Inc(A[k])
  ```

  ...should have the same effect as if it were replaced with:

  ```
  A[k] := A[k] + 1;
  ```

- ...so, meets Algol's output parameter need;

14 - examples, p. 1

- isn't pass-by-name just the same as pass-by-reference, then? NO, not always!
- Consider:

```
procedure S (el, k);
    integer el, k;
    begin
        k := 2;
        el := 0;
    end;
```

- Assume we have the following fragment of code:

```
 A[1] := 1;
 A[2] := 1;
 i := 1;
 S (A[i], i);
```

- what would happen?

15 - examples, p. 2

- for pass-by-name:
- Since the effect of pass-by-name is supposed to be as though the arguments were literally substituted for the parameters, then:

```
   S( A[i], i )
```

- should be the same as:

```
  i := 2;
  A[i] := 0;
```

- ...what is actually happening here, then?
- argument `i` is really set to 2;
- and so it is `A[2]` that gets set to 0,
- and `A[1]` is unchanged!
- upon return:

```
A[1] = 1,  i = 2,  A[2] = 0
```

16 - examples, p. 3

- IF Algol had pass-by-reference:

```
   S( A[i], i )
```

- `i` is 1 at the time of this call, so the address of `A[1]` is copied into `el` -- any changes to `el` will change `A[1]`
- the address of `i` is copied into `k` -- any changes to `k` will change `i`
- so, `k := 2;` ...sets `i` to 2,
- and `el := 0;` ...sets `A[1]` to 0
- "[MacLennan, p. 130] "The fact that in the meantime `i` has been changed to 2 has no effect on `el` since the reference to `A(i)` was computed at run time."
- upon return:

```
A[1] = 0,  i = 2,  A[2] = 1
```

17 - examples, p. 4

- and if these parameters were pass-by-value,

```
   S( A[i], i )
```

- The values of `A[1]` and and `i` would be copied into `el` and `k`;
- `el` and `k` would be changed, but NOT `A[1]` and `i`;
- after the call? NONE of `A[1]`, `A[2]`, nor `i` are changed.
- upon return:

```
A[1] = 1,  i = 1,  A[2] = 1
```

18 - Jensen's device

- Jensen's device makes explicit use of pass-by-name's distinctive behavior;
- if you want a function Sum such that x = summation of V[i] from i=1 to n
  - that's easy with pass-by-name parameters!

```
real procedure Sum (k, st, u,
                    ak);
      value st, u;
      integer k, st, u;
      real ak;
      begin
          real S;
          S := 0;
          for k := 1 step st
              until u do
              S := S + ak;
          Sum := S;
      end;
```

19 - Jensen's device, continued

- notice -- there's NO array in `Sum`!
- BUT -- consider the call:

```
x := Sum(i, 1, n, V[i])

          k   st u   ak
```

- substituting the variables in the body:

```
 S := 0;
 for i := 1 step 1 until n do
     S := S + V[i];
 Sum := S;
```

- see how `V[1]` through `V[n]` WILL indeed be changed here?
- ...but DIDN'T pass the entire array!

20 - Jensen's device, continued

Want the summation of `i` from `1` to `m`
and `j` from `1` to `n`
of `A[i, j]`?

```
x := Sum (i, 1, m, Sum (j, 1, n, A[i, j]))
```

...which becomes:

```
      S := 0;
      for i := 1 step 1 until m do
          S := S + Sum(j, 1, n, A[i, j]);
      Sum := S;
```

...and:

```
  S := 0;
  for i := 1 step 1 until m do
      S := S +   >   S := 0
                     for j := 1 step 1 to n do
                          S := S + A[i, j];
                     Sum := S;
```

- ...very flexible!

21 - to read more about how pass-by-name is implemented...

- ...see MacLennan p. 132;
  - ...NOT as awful as it sounds, but not that cheap; either;
  - using **thunks**: pieces of code that provide an address;

22 - pass-by-name trap!

- ...for example, consider this Swap routine:

```
procedure Swap(x, y);
    integer x, y;
    begin
        integer t;
        t := x;
        x := y;
        y := t;
    end;
```

- usually, all is well; BUT what if:

```
Swap (i, A[i])?
```

  - (for ex, where `i` is 1 and `A[1]` = 27)

```
t := i;       -- t set to 1
i := A[i];    -- i set to 27
A[i] := i;    -- A[27] set to 1!
```

- ... doesn't swap! doesn't exchange!

23 - pass-by-name trap, cont'd

- [MacLennan]:
- "computer scientists have shown that there is NO WAY to define a Swap procedure in Algol-60 that works for all parameters"!
- "traps such as these have led language designers to avoid pass-by-name in almost all languages designed after Algol-60"...!

24 - Ada's approach (p. 1)

- [source: pp. 286-288, MacLennan, Ch. 8]
- Ada's approach - if the idea is to permit input, output, and input/output parameters, why not just have the programmer give the intent of the parameter when declaring it?
- ...so, in Ada, parameters can be declared as `in` (input), `out` (output), or `in out` (input/output)
- then, the compiler determines how to accomplish the actual parameter passing; (and a program in which the particular parameter-passing mode used affects the output is considered to not be "legal" Ada...!)
- `in` parameters may not be assigned to;
- in Ada 83, `out` parameters were considered write-only; in Ada 95, they can be read (but presumably after they've been written to...!)

25 - Ada's approach (p. 2)

- `in` parameters that are elementary types and pointers use pass-by-value;
  - for composite types and certain other types, either pass-by-value or pass-by-reference is used, compiler's choice;
- `out` parameters use either pass-by-reference or are copied out (the result half of pass-by-value-result!), compiler's choice;
- `in out` parameters that are elementary types essentially use pass-by value-result;
  - for most composite types, either pass-by-reference or pass-by-value-result are used, compiler's choice
- see pp. 287-288 for a discussion on how the compiler can decide whether to pass a parameter by reference or by value;

26 - Parameter Passing Modes of Some Major Languages (p.1)

- sources:
  - http://74.125.155.132/search?q=cache:9JCmmH4ukVcJ:www.csie.ntu.edu.tw/~pangfeng/PL%2520notes/chap9.doc+ANS+Fortran+standard+pass+by+value+result&cd=5&hl=en&ct=clnk&gl=us
  - http://www.angelfire.com/trek/katorejas/
- **Fortran**: Pass-by-reference before F77, pass-by-value-result afterwards.
  - [katorejas:] "the language does not specify whether pass-by-reference or pass-by-value-result should be used."
- **ALGOL 60**: Pass-by-name and pass-by-value.

27 - Parameter Passing Modes of Some Major Languages (p.2)

- **C**: Pass-by-value, pass-by-reference is emulated by passing pointers.
- **C++**: Pass-by-value plus pass-by-reference with reference data type.
- **Java**: only has pass-by-value! (When an object is passed as a parameter, a reference to that object is copied -- that is *not* the same as pass-by-reference, since what is referenced can be changed, but not the reference itself...)
  - passing a Java reference by value, then, is like passing a C++ pointer by value -- it is *not* the same as passing a pointer by reference...
- **ALGOL-W**: First use of pass-by-value-result.
- **Pascal**: Pass-by-value and pass-by-reference.

28 - Parameter Passing Modes of Some Major Languages (p.3)

- **Ada**: Pass-by-value, Pass-by-reference, Pass-by-value-result, Pass-by-result (out parameters)
- **Scheme**: Pass-by-value
- **SIMULA-67**: Pass-by-name
  - [katorejas:] "Primarily because of the difficulty in implementing them, pass-by-name parameters were not carried from ALGOL 60 to any subsequent languages that became popular, other than SIMULA-67."