FORTRAN

FORTRAN Concepts/Contributions

- Binding time
- Separate compilation
- Modules
- Retention (data)
- Initialization of data objects
- Strong/weak typing (FORTRAN definitely weak!)
- Auto declaration
- Operator overloading
- Aliasing (in common blocks)
- Coercion

FORTRAN Design Considerations

- Underlying hardware
- Efficiency (time)
  - direct translation to machine ops
  - optimization
    - DO loops
    - array subscripting
- Efficiency (space)
  - equivalence
  - common
  - flat structure (no recursion)

FORTRAN & Design Principles

- Abstraction (control)
- Defense in depth (assigned, computed GOTO; DO loop)
- Structure (goto's)
- Syntactic consistency (two goto types)
- Preservation of information (DO loop)
- Zero-one-infinity (array dims, identifier length)
- Regularity (strings are second class)

FORTRAN: Interesting Problems

- Array subscripting
  - limit of three dimensions
  - limit on subscript expressions
- Parameter passing
  - reference as implemented is dangerous (expr actuals)
- Computed/Assigned GOTO's
- Syntax
  - space compression combined with no key words

    DO 10 I = 1.10

    DO 10 I = 1,10

ALGOL60

ALGOL was indeed an achievement.
It was a significant advance on most
of its successors.

--Alan Perlis

ALGOL60

- Design by committee -- US/Europe -- 1957
- Goals:
  - Machine independence
  - Improve RE: FORTRAN's established flaws
  - One standard to end "proliferation" of languages
- Model of Computation:
  - Static block structure (gave additional control over name space)
  - Recursion [multiple instances of same routine(s)] --birth of stack model

ALGOL60 Contributions

- Declaratives:
  - Named data objects
  - Declared their types
  - Determined storage in activation record
    - (or what needed to be done at runtime)
  - Bound name to that storage
  - Allowed for initialization
- Block structure: scope, visibility, hiding
  - Scope: range over which name is defined
  - Visibility: set of names that have meaning
  - Hiding: re-use of name (in new context)

c.f. FORTRAN:
- named the object
- allocated fixed memory
- provided for initialization

ALGOL60 Contributions (2)

- Static vs dynamic referencing became an issue
  - Static: meaning of non-local ref determined by static context
  - Dynamic: meaning of non-local ref determined by dynamic call chain

ALGOL60 Types

- Primitive types:
  - real, integer, boolean, strings (2nd class)
  - *no* dub precision (for machine independence)
  - *no* complex
- Constructors:
  - arrays only
    - arbitrary bounds
    - arbitrary dimensions
    - arbitrary subscripts

      Zero-one-infinity
      - includes functions and other array refs
    - dynamic sizing on block entry

ALGOL60 Types (more)

- In general, declarations were *required*
  - no auto declarations (except procedure formals, labels)
- Strong typing:
  - The only operations allowed to be performed on a data object are those defined for objects of that type
    - (one of many defs for strong typing. Others?)
- Many loopholes:
  - labels and integers
  - specifications and declarations

ALGOL60 Imperatives

- Imperatives:
  - Computational, control flow, *no I/O*
- Computational-
  - assignment much more general than FORTRAN's

    fac:= IF x <= 1 THEN 1 ELSE x*fac(x-1)
- Control flow-
  - Conditional had awkward inconsistency:

    IF ~ THEN s1 ELSE ~ -- s1 can't be cond'l

    --violates regularity

ALGOL60 Oddities

- Conditional booleans odd:

  IF IF IF a THEN b ELSE c THEN d
  ELSE f THEN g ELSE h < k
- For loop -- very general

  FOR i:= 3,7,
  11 STEP 1 UNTIl 16,
  i/2 WHILE i >= 1, 2 STEP i UNTIL 32 DO ~
  (generates: 3,7,1,12,13,14,15,16,
  8,4,2,1,2,4,8,16,32)

  --user can modify loop indices (see Knuth)
  -- violates localized cost

Switch (early case / switch statement)

```
BEGIN
   switch status= in_air, take_off,  landing, on_runway
     …
   i:= <integer value>
   goto status[i];    -- out of range treated as fall-through
     in_air: ...        }  no ordering req'mnts
     landing: ...       }  can be
     take_off: ...      }  anywhere
     …
   END

BEGIN
  switch s = L, IF j > 0 THEN M ELSE N, Q
  …
  i:= <integer value>
  goto s[i];  -- dest depends on j if i=2.
  …
END
```

Switch
- supports labeling
- violates structure & security

Designational expression

See Knuth for some gems (e.g. p. 617).

Example from Knuth: Switch

```
begin integer nn;
  switch A := B[1], B[2];
  switch B := A[G], A[2];
  integer procedure F(n, S); value n; integer n; switch S;
    begin nn := n; go to S[1]; F := nn end F;
  integer procedure G;
    begin integer n;
    n := nn; G := 0;
    nn := if n ≤ 1 then n else F(n−1, A) + F(n−2, A)
    end S;
  outreal (1, F(20, A)) end.
```

The output of this program should be 6765 (the twentieth Fibonacci number).

Name Parameter Passing

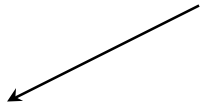
```
PROCEDURE swap (x, y);
 integer x, y;
 BEGIN INTEGER t;
   t:= x;
   x:= y;
   y:= t;
END
     swap (i,j)     -- works (x:i, y:j)
     swap (A[i],i)  -- works (x:A[i], y:i)
     swap (i,A[i])  -- doesn't work! (x:i, y:A[i])

     -- Similarly, Knuth claims you can't write a general
        successor function.
                              -- Why?
```


Jensen's Device

```
real procedure SUM (EXPR, INDEX, LB, UB); value LB, UB;
   real EXPR;  integer INDEX, LB, UB;
 begin real TEMP;  TEMP := 0;
   for INDEX := LB step 1 until UB do TEMP:= TEMP + EXPR;
   SUM:= TEMP
end proc SUM;
```

Manifest interface???



```
SUM(A[I], I, 1, 25);

 SUM (SUM(B[I,J], J, 1, N), I, 1, M)  -- How many calls to SUM?
```

Undesirable Interactions

```
BEGIN
 INTEGER max, m, n;
 READ (max);
 FOR i:= 1 UNTIL max DO
  BEGIN
   READ (m,n);
   BEGIN
    OWN REAL ARRAY a[1:m, 1:n];  --Storage?
        ...
        <operations on a>
        ...
   END
  END
END
```

- How can we deal with changing bounds while trying to keep a copy of "a" around between block entries?