ALGOL 60

Tommy McMichen and Karl Reese

History

- Preceded by ALGOL 58 - 1958
  - International Algebraic Language (IAL)
  - Developed in Germany
  - Lost out to FORTRAN
  - Introduced code blocks
- ALGOL 60 - 1960
  - A revision of ALGOL 58
  - Developed by a more international team
  - Primarily research-focused
- ALGOL W
  - Was going to be the next version of ALGOL but wound up becoming Pascal

Objectives

1. Simplify program preparation
2. Simplify program exchange
3. Enhance common programming techniques at the time

- Machine-independent
  - “Report on the Algorithmic Language ALGOL 60” defines the language
  - Translators (compilers) are required for each target architecture
- Research-oriented
  - “[W]e hope that it will become the main vehicle for communication of algorithms…”

Statements

- Common Statements
  - Arithmetic Assignment
  - Boolean Assignment
- Control Flow
  - Go To
  - For
  - Conditional (If - Then - Else)
- Common I/O (Not included in ALGOL 60)
  - Read
  - Punch
  - Carriage Return
- Compound
  - begin … end

Recursive Definition

- All expressions are made of other expressions
  - Example: Conditional Statement

    if **B** then $\mathbf{S_1}$ else $\mathbf{S_2}$

    where **B** is a Boolean expression and $\mathbf{S_1}$ and $\mathbf{S_2}$ are statements
  - ALGOL translators (compilers) can exploit this recursive nature

Types

- Explicit typing: real, integer, boolean
  - `integer b;`
- Arrays
  - Real by default, but can specify type
  - Can also be multi-dimensional

  -
    ```
    array a[-1 : +2];
    boolean array b[1:10, 3:4];
    ```
- Strings
  - Not strictly part of ALGOL 60 but can be useful for I/O
  - `'hedgehog'`
- Switches
  - Like arrays of goto labels

  -
    ```
    switch s := A, B, C, D;
    go to s[2];
    ```

Comments

```
comment The text between the comment symbol and the next semicolon constitutes
a comment. ;

if (i = 43) then
begin
    …
end Comments starting after end can be ended by end, else, or a semicolon.
else if (i = 42) then
    ...
```

For Loops

- Provided as a convenience to the programmer

`for i := 1 step 1 until 10 do`
`    punch i;`
	`for i := 1, 2, 3 do`
	`    punch i;`
		`a := -1;`
		`for i := 1 while (i > a) do`
		`begin`
		`    punch i;`
		`    a := a + 1;`
		`end;`

1 2 3 4 5 6 7 8 9 10	1 2 3	1 1

Blocks

- Introduce scope to program
  - Variables defined in a block are local to that block
    - Cannot be used in external blocks
    - Cease to exist once program leaves the block
      - Unless they are declared with the ***own*** keyword
- Can be nested

```
begin real a, b;
    a := 3.14;
    begin integer a;
        a := 42;
    end;
end
```

```
A:  begin own real y;
        …
        y := 7;
    end
    …
B:  go to A;
```

Procedures and Functions

- Typed Parameters
  - Optional but recommended
- Value Parameters
  - To be pre-computed at the start of the function
- Functions
  - Procedures with return values, can recurse

```
procedure A(x, y): value x, y;
    punch x + y;

A(a+b, c);
```

```
real procedure fact(n): real n;
    if n = 1 then fact := 1
    else fact := n x fact(n-1);

fact(4);
```

Procedures and Functions, cont’d

- Procedure calls can be handled inline rather than jumping
    - Formal parameters are copied from the call into the procedure
        - They are marked for evaluation if they are value parameters
    - The procedure is then copied into the local scope with further alpha-substitutions as necessary
- Recursion and own-variables make this a little more complicated
    - “Most translators presently under construction will not handle recursive procedures”
- This is ultimately a translator-specific choice
    - Runtime vs. Program Size

Questions?