ALGOL-60 and ALGOL-68

ALGOL-60 was developed cooperatively between the US and Europe, with the goal of standardizing an international general-purpose programming language. Its primary design goal was *portability*. Since I/O devices were nowhere close to standization, ALGOL depended upon external libraries for device drivers. That is, ALGOL had no `read` or `print` statements.

The language definition is 15 pages long, exemplifying brevity and clarity through the use of BNF descriptions. ALGOL-60 introduced several central programming tools, in particular

Hierarchical structure evolved into *structured programming:*

Typed procedures (functions)

Stack as central runtime data structure
managed by block structuring; dynamic array sizing

Compound statements (regularity)
any valid operation can be written wherever variables can be written
block structure and nested scoping
implicit inheritance of accessible (within scope) variables

No magic values
name lengths and array size and dimension are not restricted

Strong typing
all forms are typed without ambiguity

Generalized control structures
`while`, `until`, `for`, recursion

Free Format
layout of the program is not specified by the language; reserved words are sacred

Implications of Block Structuring

Nested blocks introduced the issue of *variable scoping*.

Blocks enhance *modularity* and *maintenance* of large programs.

Blocks permit *efficient storage* management of stacks.

Symbol tables can be *factored* into smaller units for each block.
Symbol tables are accessible to the user, which compromises portability.

Blocks need to be *bracketed* to delineate start and end points. `Begin` and `end` are introduced as generic brackets. In ALGOL, `begin-end` brackets both group statements and delimit blocks. This undermines orthogonality.

Dynamic and Static Scoping

Static scoping: procedure definitions are called in the defining context at compile-time. Variable bindings can always be determined by reading the source code.

Dynamic scoping: procedure definitions are called in the calling environment at run-time. Variable bindings depend on dynamic context. *Example:*

```
begin integer m;        ;outer block
  procedure P;
    m := 1;
    begin integer m;
      m := 2;
      P                       ;first call to P
    end;
  P                           ;second call to P
end
```

In *static scoping*, the assignment `m:=1` refers to the variable `m` in the outer block. Static means that no matter when a particular procedure is called, the binding context of its variables does not change. Thus both calls to procedure `P` use `m=1`.

In *dynamic scoping*, the assignment to `m` depends upon the calling context. The first call to `P` is in the context of `m=2`, thus it uses that value. The second call to `P` is in the outer context where `m=1`. [Note that the value `m=2` is changed to `m=1` immediately upon entering `P`.]

Parameter Passing

Pass by value: the *actual value* of the variable being passed to a procedure is copied into the formal parameter of the procedure. Secure, but inefficient for arrays. For input variables only.

Pass by name: the *name* of the variable being passed to a procedure is copied into the formal parameter of the procedure. Powerful, but dangerous and expensive.

Each type of parameter passing differs in when it looks at the value of the variable being passed (the actual) into the procedure (the formal parameter):

Pass by value: When a procedure is called, the formal is bound to the value of the actual as a snapshot. Later changes in the actual will not be seen by the procedure. Early inspection time.

Pass by reference: When a procedure is called, the formal is bound to a reference to the actual. The reference cannot change (i.e. the location of the actual can not be changed), but the value it refers to can change.

Pass by name: When a procedure is called, the formal is bound to special address-returning function (a thunk). Although the thunk does not change, the address it returns and the value in the address location can vary. Very late inspection time.