FORTRAN

FORTRAN was the first language with the design goal of *efficient performance*. Consequently, the constructs of the language are designed to accommodate a specific machine architecture. FORTRAN was also essentially a numerical processing language for scientific computation. The new features introduces by the language include:

Subprograms

- modularity
- communication through parameter binding
- procedural abstraction
- libraries

Two-part programs

- declarations
  - non-executable, compile-time directives
    - memory allotment
    - names for the memory spaces
    - initial contents of memory
- instructions
  - executable, runtime
    - computation through assignment (arithmetic and move ops)
    - control flow through IF and DO
    - input/output

Several processing stages (for efficiency)

- compile
  - relocatable object code (subprograms may move)
- link
  - thread libraries and external references
- load
  - absolute memory format
- execute
  - program in memory controls computer

Imperative programming

- flow and control governed by programmable control logic

GOTO

- single low-level transfer of control
- confusing mental model
- static and dynamic models don’t match

DO loop

initialization, iteration, and return all directly controlled within DO

Coercion

allow mixed operations

Limited arrays

optimize memory
use array index as memory address (rather than computing new addresses)

Implications of Subprograms

```
SUBROUTINE  <name>  <formal parameters>
```

Inefficient, naive invocation:

Substitute the subroutine for its name in the main calling body of code, and substitute the calling values for the formal parameters

Pass by Reference (FORTRAN’s solution)

Substitute the *location* of the subroutine in memory
for its name in the main calling body of code

difficult to understand dynamic behavior
security risk when locations can be accessed

Pass by Value (preferred)

Substitute values for parameters in subroutine
Run subroutine in place
Return result to calling context

requires activation record to keep track of bookkeeping

Activation Records

parameter bindings	new values passed to the subroutine
resume address	place to return control when subroutine is done
dynamic link	location of caller’s activation record, for returning results
temporary storage	for subroutine bookkeeping

Subprogram Invocation

To CALL

1. Place parameter binding values in callee's activation record.
2. Save caller state and resume address in caller's activation record.
3. Place pointer to caller's activation record in callee's activation record
4. Enter subprogram (callee's) first instruction

To RETURN

5. Transfer to callee's resume address.
6. When caller gains control, restore caller state.
7. When subroutine has return values (i.e. callee is a function),
   place return values in caller's activation record.

Name Structures (Environments, Symbol Tables)

Environments define *context* and *meaning*.

Sample name space:

name	***type***	***location***	***value***
IF	reserved	0247	<control>
i	integer	0248	3
res	list	0249	(a b c)
my-plus	function	0250	<body>

Each subprogram has its own name space for local variables.

Subprogram names must be global.

In FORTRAN, COMMON blocks declare shared data. This *aliasing* makes code maintenance confusing.