Introduction, or Why Bother With This Stuff, Anyway?

- Increased capacity to express ideas.
- Improved background for choosing languages.
- Increased ability to learn languages.
- Understanding of significance of implementation.
- Ability to design new languages.
- Overall advancement of computing.

Language Evaluation Criteria

- **Readability & Writability**
  - **Simplicity**
    - small # of basic components
      (subsets a poor solution!)
    - one syntax: one meaning

      Counter-example in C: four ways to increment x:

      ```
      x++;
      x=x+1;
      x+=1;
      ++x;
      ```

      In FORTRAN, two meanings for:

      ```
      Y=SUM(I,J)
      ```

      -- array reference

      -- function call

Language Evaluation Criteria

- **Orthogonality**

  => Any composition of basic primitives is allowed

- need small set of primitives, ways to combine them
- Pascal not very orthogonal.

  Functions can't return structured types.

  Type of formal parameter must be stated in function/ procedure heading unless parameter is a function or procedure.

  Enumerated types can't be read or written.

  etc...

- Non-orthogonality is often to simplify implementation.
- LISP is much more orthogonal than Pascal.

Language Evaluation Criteria

- **Control Statements/Constructs**

  Importance and desirability of various control mechanisms with the language.

- **Data Types**

  Rich set of data types makes programs much easier to writ understand. Provides abstraction.

- **Syntax**

  Matters more than you think!

  Identifier length, reserved words, layout, etc.

- **Abstraction**

  Must be able to hide details, or complexity is too great.

  process abstraction

  data abstraction

Language Evaluation Criteria

- **Reliability**
  - **Definition: performs to specifications under all conditions**
  - **Impact from:**

    type checking (or lack thereof)

    compile-time *best*

    runtime *good*

    exception handling

    Special language features to help intercept and handle unusual situations. No magic.

    Somewhat controversial.

    aliasing

    Two or more names for same memory cell.

```
var p,q: ^int;
begin
  new(p);
  q:=p;
```

Language Evaluation Criteria

- **Cost**
  - **More than just runtime!**

    time to train programers

    *program development time

    compile time

    runtime

    *maintenance time (>50%!)

  - *** functions of writability and readability**

    => most important

Influences on Language Design

- (1) Computer architecture
- (2) Programming methodologies
- Historically more of (1), moving toward (2). Getting higher and higher-level:

machine language

↓

assembly

↓

FORTRAN

↓

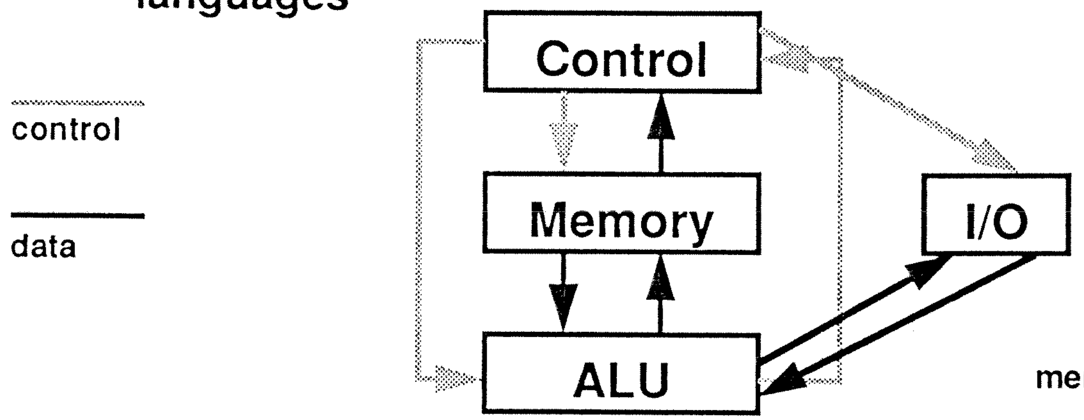
Algol-like languages

↓

"very high-level" languages:
functional, logic, etc.

The von Neumann Architecture

- **John von Neumann at Princeton**
  - **late 1940's**
  - **has influenced the design of most programming languages**



memory
↕
processo

- **Fetch-execute cycle:**
  - **Control fetches next instr from memory**
  - **Control decodes instr**
  - **Execution**

    data from memory to ALU, or

    data from ALU to memory, or

    I/O

The von Neumann Architecture

- **The von Neumann architecture is reflected in traditional programming languages in two ways:**
  - **sequential, step-by-step execution of instructions**
  - **modifiable variables -- "cubbyholes" in memory**
- **These languages became popular and drove further architectural designs. Vicious circle . . . other language designs didn't have much chance until recently.**

Translation and Interpretation

- **We could build a special machine to execute each language directly, but this is impractical. So how to get a program in a high-level language down to machine code?**

- **Interpretation**

  An interpreter takes statements of a program one at a time and executes them directly as follows:

  Get next statement

  Determine actions

  Perform actions

  Repeat

  (*look familiar?*)

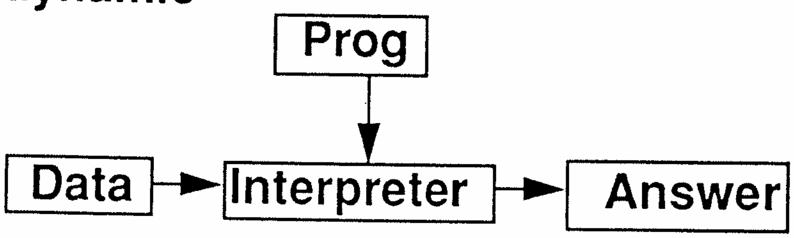
Notes on Interpretation:

- data is provided to the interpreter as required

- one high-level instruction (HLI) => one sequence of machine-level instructions

- redetermine actions each time HLI is encountered

- highly dynamic



Translation

- **Translation**
  - **A translator takes a program in language A and produces an equivalent program in language B. If B is "closer" to machine code than A, it's called a compiler.**
- **Notes on translation:**
  - **high-level program --> machine-level prog**

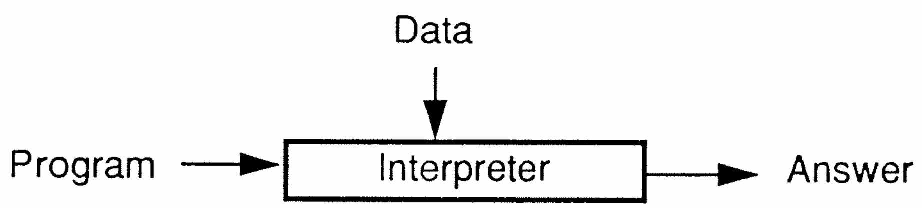
    (*not* HLI-->MLI)
  - **parsing + code generation**
  - **decode each statement once**

    => saves time
  - **store expanded version of program**

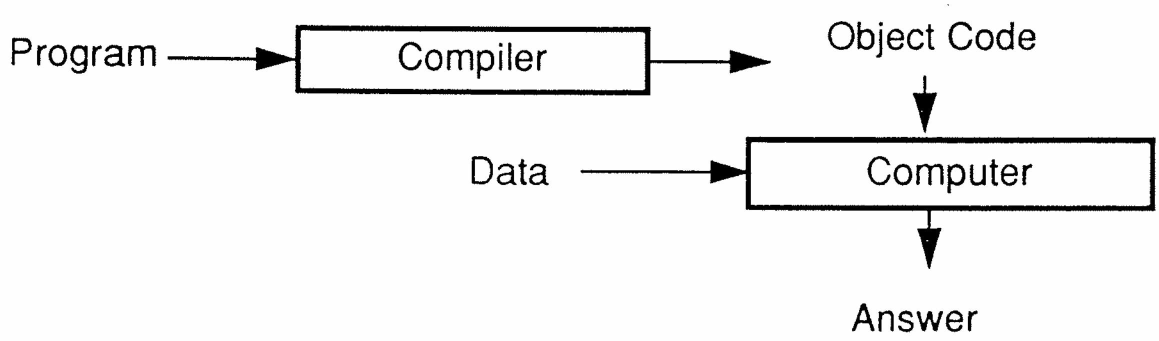
    => costs space

Interpretation vs. Translation

- **Interpretation**



- **Translation**



  - **Combine translation and interpretation by substituting Interpreter for Computer above.**

A Brief History of Programming Languages

- **Wegner, "Prog. Langs-The First 25 Years", IEEE Transactions on Computers, 12/76, 1207-25**

- ***1950's "Discovery and description"***
  - **assembly**
  - **FORTRAN, ALGOL60, COBOL, LISP**
  - **basic implementation techniques**

    symbol tables

    stack evaluation of arithmetic

    activation records

    garbage collection
  - **languages as tools**
  - **late 1950's: first compilers (Hopper, etc.)**

    grammars and automata (Chomsky and Miller)

A Brief History of Programming Languages

- ***<u>1960's</u>* "Elaboration and analysis"**
  - **theories of programming languages**
  - **more formal development**
    - formal languages
    - automata
    - formal semantics
    - verification
  - **bigger, more complex languages**
  - **PL/I, Simula, ALGOL68**
  - **late 1960's: theoretical work on compilers, program optimization**

A Brief History of Programming Languages

- ***1970's* "Technology"**
  - **practical issues**
  - **applications of computer science**
  - **hardware cheaper, faster**
  - **software complexity increased**
  - **programming methodologies**
    - structured programming
    - program verification
  - **Pascal, C, Modula, Clu**

A Brief History of Programming Languages

- ***1980's***
  - **parallel hardware => parallel language**
  - **very high-level languages**
    functional
  - **logic**
  - **(object-oriented)**

Specific Milestones

- *1944*: **EDVAC (Electronic Discrete Variable Automatic Calculator) Report (von Neumann)**
  - **first description of a stored-program computer**
- *1950*: **First Assemblers**
- *1954-57*: **FORTRAN ("FORmula TRANslating system")**
  - **Backus et al @ IBM**
  - **Goals:**

    efficiency -- less than twice as slow as assembler

    solve economic problem -- design, coding, debugging too expensive in assembly
  - **elegance of design secondary**
  - **Versions I, II, III, IV**
  - **introduced separate compilation with II because programs were getting too large to compile withou hardware errors**
  - **"An existence proof for higher-level languages..."**

Specific Milestones

- *1958-60*: **ALGOL 60 ("ALGOrithmic Language")**
  - **by committee, including Backus**
  - **Goals:**

    elegant, universal language (FORTRAN was for IBM)

    standard mathematical notation

    major contributions:

      BNF

      block structure

      recursion

      call-by-value/name

      stack model of evaluation

      semi-dynamic arrays

      but no formatted I/O -- too machine-dependent

- *1956-62*: **LISP ("LISt Processing")**
  - **McCarthy @ MIT**
  - **for symbolic computation in AI**
  - **free of von Neumann concepts**
  - **(roughly) based on lambda-calculus**

Specific Milestones

- *1956-62*: APL ("A Programming Language")
  - Iverson @ Harvard
  - array processing
  - functional flavor, fairly non-von Neumann
  - didn't catch on until 1970's
- 1960: COBOL ("COmmon Business Oriented Language")
  - at U Penn by representatives of computer manufacturers
  - alienated from CS community
    - developed by commercial community; didn't ask CS'ers
    - no interest in scientific or research implications
    - no BNF definition
    - no good books
    - commercial applications thought trivial by CS'ers
  - main contribution: file/record structure
  - syntax wordy, English-like
  - very slow at first, but survived because use m by DoD
  - ref: Schneiderman, *Annals of the History of Computing*, 10/85

Specific Milestones

- *1960's*: BASIC
  - Kemeny and Kurtz @ Dartmouth
  - for teaching
  - access through terminals
  - novel idea: user time more important than machine time!
  - commercial success a surprise -- intended for their students
  - no real contributions

- *1962-67*: SNOBOL4 ("StriNg Oriented symBOlic Language")
  - Griswold @ Bell Labs
  - string processing
  - introduced pattern-matching

- *1964-69*: PL/I ("Programming Language I")
  - by committee @ IBM
  - tried to unify commercial and scientific features
  - very large; programmers learn a subset

Specific Milestones

- *1963-68*: ALGOL68
  - by committee
  - small number of orthogonal constructs
  - hard to learn -- too general and too flexible
  - poor implementations/documentation
- *1967-71*: Pascal
  - Wirth
  - small, simple -- for teaching
  - structured programming, fairly rich data structures
- *~1973*: C
  - Kernighan and Ritchie @ Bell Labs
  - low level, for systems programming
  - fairly small, fast
  - hard to read and maintain

Specific Milestones

- *mid 1970's*: Modula-2
  - Niklaus Wirth
  - Pascal and modules
  - better for systems programming and large projects
- *mid 1970's*: PROLOG ("PROgramming in LOGic")
  - Kowalski and Colmerauer @ Edinburgh and Marseilles
  - non-von Neumann, based on first-order logic (but impure)
  - most applications in AI
  - Japanese 5th generation computing project chose it
- mid 1970's: SMALLTALK
  - Xerox
  - object-oriented: shift in focus
  - not just a language; a whole system

Specific Milestones

- *mid 1970's - 80*: **Ada (after Ada Augusta, daught of Lord Byron, associate of Babbage -- "the first programmer")**
  - **DoD**
  - **requirements developed slowly:**
    - Strawman
    - Woodman
    - Tinman
    - Ironman
    - Steelman
  - **design contract won by CII-Honeywell Bull (Jean Ichbiah)**
  - **based on Pascal**
  - **large, complex**
  - **features:**
    - packages
    - tasks
    - real-time capabilities
    - exception handling

Specific Milestones

- *1980's*: C++
  - Bjorne Stroustrup
  - C + classes
  - OOP in a popular language
- *1980's*: Hope, Miranda, LML, Haskell
  - purely functional
  - based on lambda-calculus
  - higher-order functions, pattern matching, type inferencing
  - good for parallel machines?