CSC330 Summer 2005
PROGRAMMING LANGUAGES

George Tzanetakis
gtzan@cs.uvic.ca
http://www.cs.uvic.ca/~gtzan

Course Administration

- Reading UVIC emails is required
- Emphasis on work not inspiration
  - Pace yourself (work 2-3 hours/lecture)
- Attendance is critical and lectures are incremental
- Grading, copying
  - Oral interviews for programming assignments
  - Open book policy for midterm/final

A topic for discussion

In CACM The next 1000 yrs , Vol 44 (3) with topics such as Digital Immortality, Virtual Beings, Cyborgs etc.

“Computing's central challenge: “How not to make a mess of it” has not been met. On the contrary, most of our systems are much more complicated than can be considered healthy, and are too messy and chaotic to be used in comfort and confidence. The average customer of the computing industry has been served so poorly that he expects his system to crash all the time, and we witness a massive worlwide distribution of bug-ridden software for which we should be deeply ashamed.” - E. Dijkstra

Programming Languages Poll

- “The only way you can learn a programming language is by writing programs in it” - B. Kernighan & D. Ritchie
- How many PLs have you programmed in ?
- How many PLs do you know ?
- What's the most difficult program you have written ?

What is a Programming Language ?

- Is English a PL ?
- Is Adobe photoshop a PL ?
- Is Excel a PL ?
- Is Matlab a PL ?

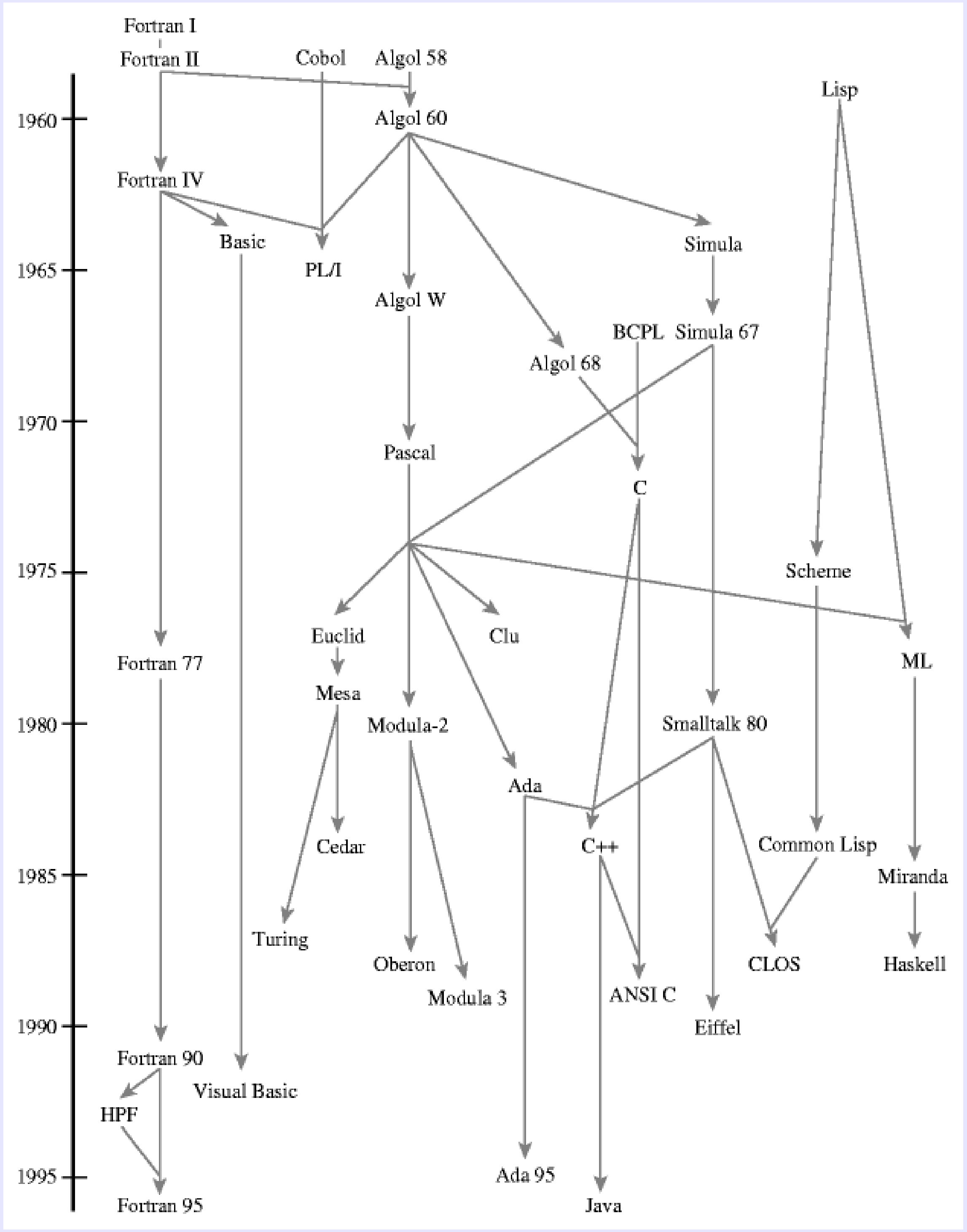
What is a programming language ?

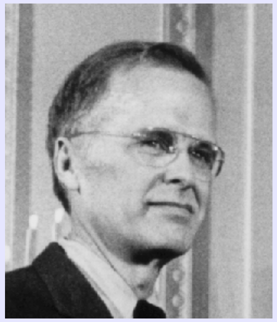
A programming language is a system of notation for describing computations. A useful programming language must therefore be suited for both description (i.e., for human writers and readers of programs) and for computation (i.e., for efficient implementation on computers). But human beings and computers are so different that it is difficult to find notational devices that are well suited to the capabilities of both. -
R. Tennant (Principles of Programming Languages, Prentice Hall, 1981)

One doesn't really understand the bones of a language until one has tried to design one. -
J.R.R Tolkien when asked why spend years designing "High Elvish":

History of PLs

The grandparents:
Fortan 50s
Cobol 50s
Algol 60s
Lisp 50s





FORTRAN (Formula TRANslator)

- John Backus & team in 1954
- Goals: scientific computing, efficiency
- Important concepts
  - High-level programming language (X=Y*Z+Q)
  - Translator (what we call compiler)
  - Machine-independent programs
  - Floating point numbers

"We did not regard language design as a difficult problem, merely a simple prelude to the real problem: designing a compiler that could produce efficient programs." J. Backus

Cobol
(COmmon Business Oriented Language)

- Grace Hopper 1950s designed FlowMatic which led to Cobol in 1959
- Business applications
  - Record structure
  - Separation of data structures from execution
  - Emphasis on readability but very WORDY
  - Versatile formating
  - “I had a running compiler and no one would touch it. They said computers could only do arithmetic”

ALGOL

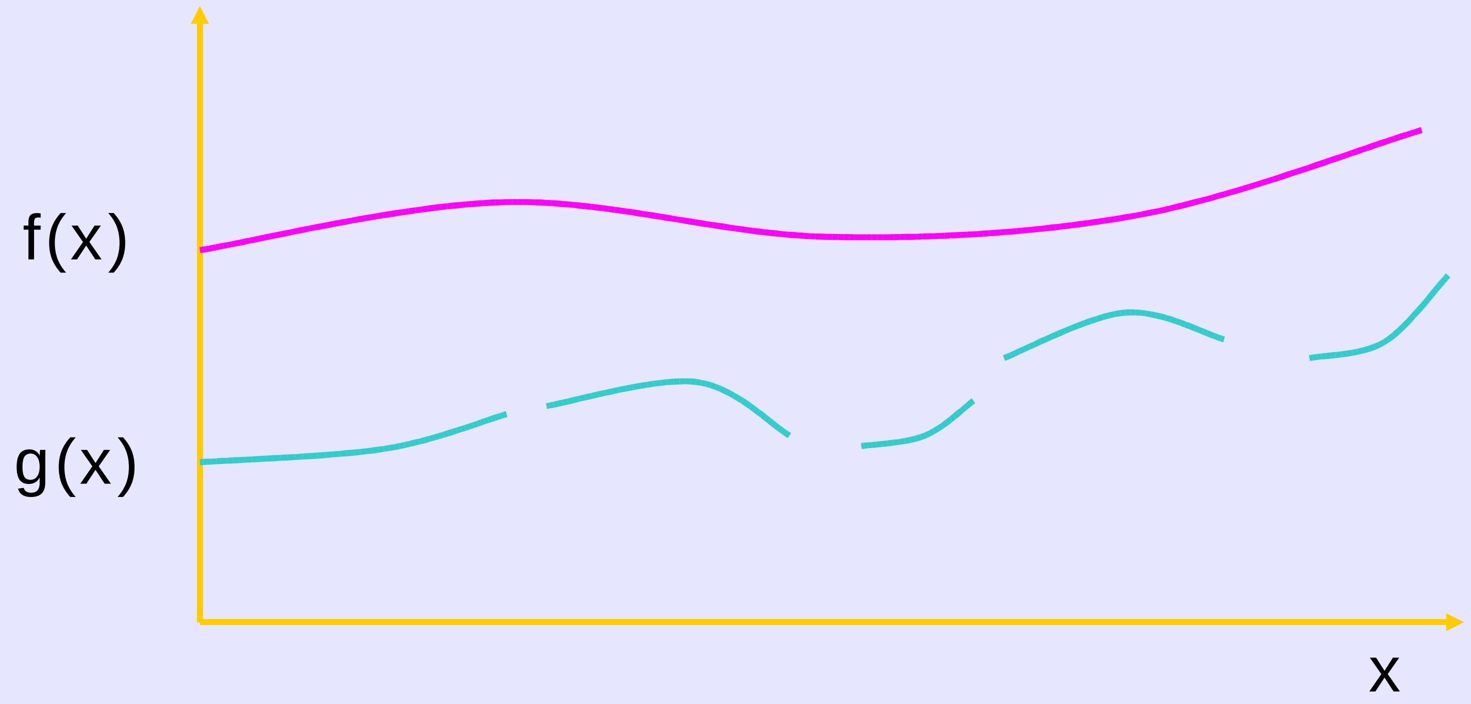
- Designed by an international committee to be a universal language
- Nested structure of environments & control
- E-BNF (Backus Naur Form) syntax specification
- Never really used except for publishing algorithms
- Huge influences on future languages

Algol 68 Minority Report

We regard the current Report on Algorithmic Language ALGOL 68 as the fruit of an effort to apply a methodology for language definition to a newly designed programming language. We regard the effort as an experiment and professional honesty compels us to state that in our considered opinion we judge the experiment to be a failure in both respects. The failure of the description methodology is most readily demonstrated by the sheer size of the Report in which, as stated on many occasions by the authors, "every word and every symbol matters" and by the extreme difficulty of achieving correctness.
(Dijkstra, Hoare and others)

No proper program contains an indication which as an operator-applied occurrence identifies an operator-defining occurrence which as an indication applied occurrence identifies an indication-defining occurrence different from the one identified by the given indication as an indication-applied occurrence. - ALGOL 68 Report

Partial & total functions



Programs are partial functions:
1) partial ops (division by zero)
2) non-term iteration

Total function: f(x) has a value for every x
Partial function: g(x) doesn't have a value for every x

Graph of f: = {<x,y> | y = f(x)}
set of ordered pairs

Computability

- Function is computable if some program P computes it: For any input x, the program P halts with output f(x)
- Terminology
  - Partial recursive functions = partial functions (int -> int) that are computable

Halting problem

- Decide whether program halts on input
  - Given program P and input x to P:

  Halt(P,x) = { yes if P(x) halts, no otherwise}
- Clarifications:
  - Assume program P requires one string input
  - Write P(x) for output of P when run with input x
  - Program P is string input to *Halt*
  - FACT: there is no program for *Halt*

Unsolvability of halting problem

- Suppose P solves variant of halting problem:

  on input Q P(Q) = {yes if Q halts, no otherwise}
- Build program D such that:

  D(Q) = {run forever if Q(Q) halts, otherwise halt}
- D(D)
  - If D(D) halts, then D(D) runs forever
  - If D(D) runs forever, then D(D) halts
  - <u>**CONTRADICTION**</u>

Main points about computability

- Some functions are computable, some are not
  - Halting problem
- Programming language implementation
  - Can report error if program is undefined due to error in operation
  - Cannot report error if program will not terminate

Things to think about

-
- Persistance of established technologies
- Interpreters, compilers, environments
- Compile-time vs run-time
- Writing vs building vs growing a program
- READ CHAPTERS 1,2