Introduction to Programming Languages and Compilers
Prof. Richard Fateman

CS164 Fall 2005

9:30-11AM

Room 22 Warren Hall

Tuesday / Thursday

Aministrivia

- Course home page:
  http://www-inst.eecs.berkeley.edu/~cs164
- Course newsgroup: ucb.class.cs164 /note security issues
- Pick up a class account here or at discussion section. You can negotiate a change in section if there is room. (TA: David Bindel)
- NEXT LECTURE and thereafter: Meet in 306 Soda Hall ("HP auditorium") [unless there are objections?]

Will CS164 be the same as last or next semester?

- **No.**
  - We are using a different text
  - We are using a different project
  - We will write programs using **Common Lisp**, not Java or C++
- **Yes.**
  - Topics covered will be similar.
  - Sequence of topics will be similar.
  - Slides (like this) will be similar. (They've evolved over years)
  - Target languages COOL and MiniJava are not THAT different.

Should you wait 'til next semester? Or later?

- It's your call. Professors Bodik and Necula will use the "COOL" language. We will use a simpler language (MiniJava).
- Professor Hilfinger has had students implementing much more elaborate languages..
- We will build upon what you learned in CS61A, so if you thought Scheme was neat and want to understand that stuff better, stay here.

Why Should You Study Compilers/ Programming Languages?

- NOTATION MAKES THOUGHT POSSIBLE

5000 years ago: Babylonian clay tablet (base 60 numbers)

500 years ago

- the modern + sign appeared at the end of the 1400s.
- This illustration from 1579 is something that looks almost modern, particularly when you see it is written in English, until you realize that those funny squiggles aren't x's-- they're special non-letter characters that represent different powers of the variable x.

more info..

- [wolfram: MathML conference]

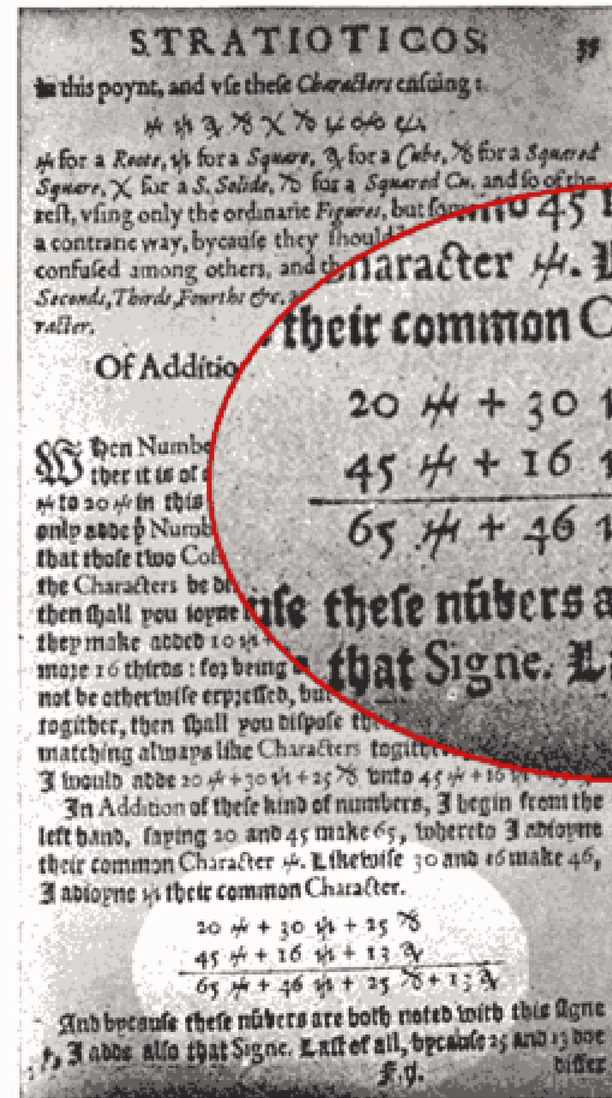

FIG. 76.—Leonard and Thomas Digges, *Stratioticos* (1579), p. 35, showing the unknown and its powers to x^6.

330 years ago (1675: Integral Calculus)

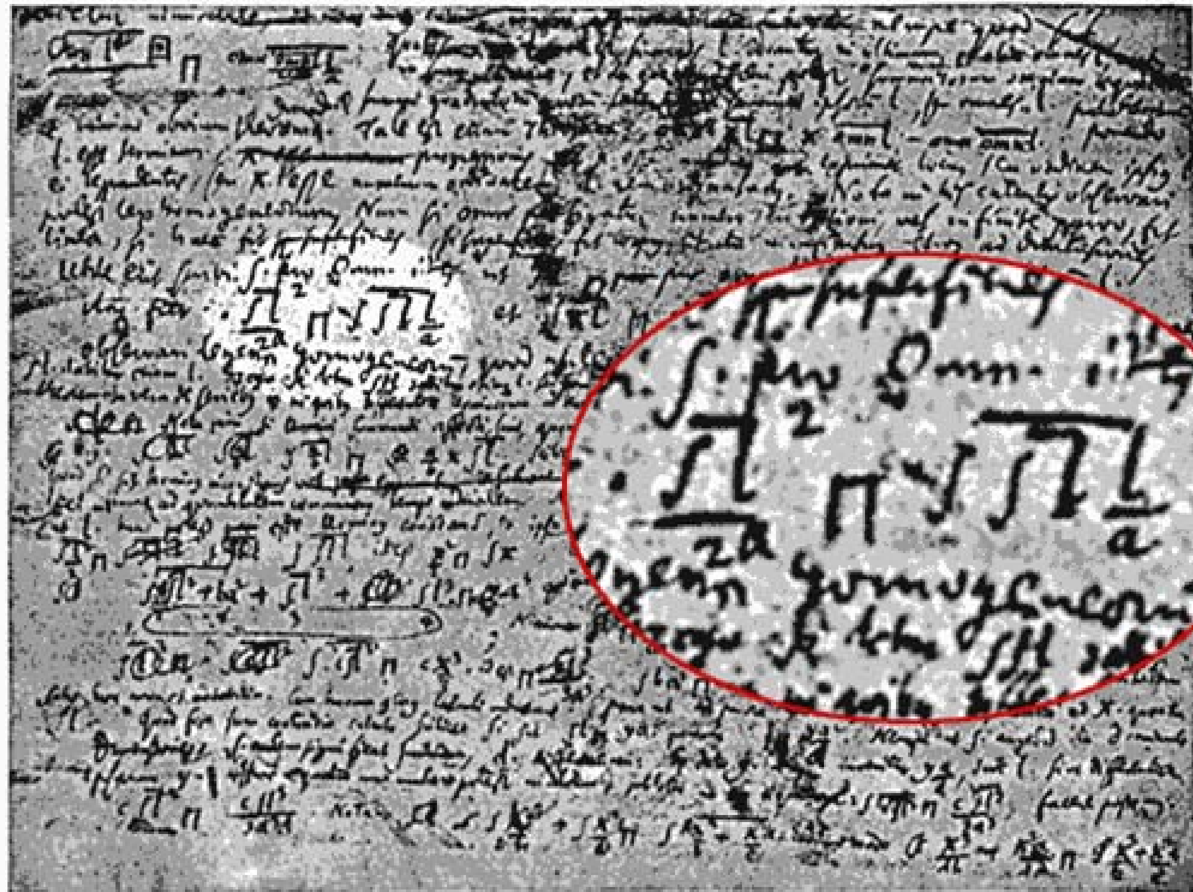

FIG. 124.—Facsimile of manuscript of Leibniz, dated Oct. 29, 1675, in which his sign of integration first appears. (Taken from C. I. Gerhardt's *Briefwechsel von G. W. Leibniz mit Mathematikern* [1899].)

100 years ago Russell/Whitehead

Principia Mathematica
1910

SECTION D] THE PRODUCT OF TWO RELATION-NUMBERS 487

***184·15.** $\vdash . \mu \dot{\times} \nu \in \mathrm{NR}$

***184·16.** $\vdash :. \mu \dot{\times} \nu = 0_r . \equiv : \mu, \nu \in \mathrm{NR} - \iota\text{‘}\Lambda : \mu = 0_r . \vee . \nu = 0_r$

***184·2.** $\vdash :. \mathrm{Mult\ ax} . \supset : P \in \mathrm{Nr}\text{‘}R . C\text{‘}P \subset \mathrm{Nr}\text{‘}S . \supset . \Sigma\mathrm{Nr}\text{‘}P = \mathrm{Nr}\text{‘}R \dot{\times} \mathrm{Nr}\text{‘}S$
[*183·26 . *184·13]

***184·21.** $\vdash :. \mathrm{Mult\ ax} . \supset : \mu, \nu \in \mathrm{NR} . \nu \neq \Lambda . P \in \mu . C\text{‘}P \subset \nu . \supset . \Sigma\mathrm{Nr}\text{‘}P = \mu \dot{\times} \nu$

Dem.

$\vdash . *152·45 . \supset \vdash : \mu \in \mathrm{NR} . P \in \mu . \supset . \mu = \mathrm{Nr}\text{‘}P$ (1)

$\vdash . *152·45 . \supset \vdash : \nu \in \mathrm{NR} . C\text{‘}P \subset \nu . S \in C\text{‘}P . \supset . \nu = \mathrm{Nr}\text{‘}S . C\text{‘}P \subset \mathrm{Nr}\text{‘}S$ (2)

$\vdash . (1) . (2) . *184·2 . \supset$

$\vdash :. \mathrm{Mult\ ax} . \supset : \mu, \nu \in \mathrm{NR} . P \in \mu . C\text{‘}P \subset \nu . S \in C\text{‘}P . \supset .$
$\Sigma\mathrm{Nr}\text{‘}P = \mathrm{Nr}\text{‘}P \dot{\times} \mathrm{Nr}\text{‘}S . \mu = \mathrm{Nr}\text{‘}P . \nu = \mathrm{Nr}\text{‘}S .$

[*13·13] $\supset . \Sigma\mathrm{Nr}\text{‘}P = \mu \dot{\times} \nu$ (3)

$\vdash . (3) . *10·11·21·23 . \supset$

$\vdash :. \mathrm{Mult\ ax} . \supset : \mu, \nu \in \mathrm{NR} . P \in \mu . C\text{‘}P \subset \nu . \exists ! C\text{‘}P . \supset . \Sigma\mathrm{Nr}\text{‘}P = \mu \dot{\times} \nu$ (4)

$\vdash . *183·2 . *162·4 . \supset \vdash : P = \dot{\Lambda} . \supset . \Sigma\mathrm{Nr}\text{‘}P = 0_r$ (5)

$\vdash . *153·16 . \mathrm{Transp} . \supset \vdash :. \mu \in \mathrm{NR} . P \in \mu . P = \dot{\Lambda} . \supset : \mu = 0_r :$

[*184·16] $\supset : \nu \in \mathrm{NR} - \iota\text{‘}\Lambda . \supset . \mu \dot{\times} \nu = 0_r$ (6)

$\vdash . (5) . (6) . \supset \vdash : \mu, \nu \in \mathrm{NR} . \nu \neq \Lambda . P \in \mu . C\text{‘}P \subset \nu . P = \dot{\Lambda} . \supset .$
$\Sigma\mathrm{Nr}\text{‘}P = \mu \dot{\times} \nu$ (7)

$\vdash . (4) . (7) . \supset \vdash . \mathrm{Prop}$

***184·3.** $\vdash . (\mathrm{Nr}\text{‘}P \dot{\times} \mathrm{Nr}\text{‘}Q) \dot{\times} \mathrm{Nr}\text{‘}R = \mathrm{Nr}\text{‘}P \dot{\times} (\mathrm{Nr}\text{‘}Q \dot{\times} \mathrm{Nr}\text{‘}R) = \mathrm{Nr}\text{‘}(P \times Q \times R)$

Dem.

$\vdash . *184·13 . \supset \vdash . (\mathrm{Nr}\text{‘}P \dot{\times} \mathrm{Nr}\text{‘}Q) \dot{\times} \mathrm{Nr}\text{‘}R = \mathrm{Nr}\text{‘}(P \times Q) \dot{\times} \mathrm{Nr}\text{‘}R$

[*184·13] $= \mathrm{Nr}\text{‘}(P \times Q \times R)$

[*166·42] $= \mathrm{Nr}\text{‘}\{P \times (Q \times R)\}$

[*184·13] $= \mathrm{Nr}\text{‘}P \dot{\times} (\mathrm{Nr}\text{‘}Q \dot{\times} \mathrm{Nr}\text{‘}R) . \supset \vdash . \mathrm{Prop}$

***184·31.** $\vdash . (\mu \dot{\times} \nu) \dot{\times} \varpi = \mu \dot{\times} (\nu \dot{\times} \varpi)$

Dem.

$\vdash . *184·111 . \supset \vdash : \sim(\mu, \nu, \varpi \in \mathrm{N_0R}) . \supset . (\mu \dot{\times} \nu) \dot{\times} \varpi = \Lambda . \mu \dot{\times} (\nu \dot{\times} \varpi) = \Lambda$ (1)

$\vdash . *155·2 . \supset \vdash : \mu, \nu, \varpi \in \mathrm{N_0R} . \supset .$
$(\exists P, Q, R) . \mu = \mathrm{N_0r}\text{‘}P . \nu = \mathrm{N_0r}\text{‘}Q . \varpi = \mathrm{N_0r}\text{‘}R$ (2)

$\vdash . *184·13 . \supset$

$\vdash : \mu = \mathrm{N_0r}\text{‘}P . \nu = \mathrm{N_0r}\text{‘}Q . \varpi = \mathrm{N_0r}\text{‘}R . \supset . (\mu \dot{\times} \nu) \dot{\times} \varpi = \mathrm{Nr}\text{‘}\{(P \times Q) \times R\}$

[*184·3] $= \mathrm{Nr}\text{‘}\{P \times (Q \times R)\}$

[*184·13] $= \mu \dot{\times} (\nu \dot{\times} \varpi)$ (3)

$\vdash . (2) . (3) . \supset \vdash : \mu, \nu, \varpi \in \mathrm{N_0R} . \supset . (\mu \dot{\times} \nu) \dot{\times} \varpi = \mu \dot{\times} (\nu \dot{\times} \varpi)$ (4)

$\vdash . (1) . (4) . \supset \vdash . \mathrm{Prop}$

***184·32.** $\mu \dot{\times} \nu \dot{\times} \varpi = (\mu \dot{\times} \nu) \dot{\times} \varpi$ Df

Iverson's APL (1960)

```
    ∇ Age Life Gen; a; b; c; h; k
St:
    mat ← mat ≠ 0
    a ← ∨/ ,mat
    ⍞ ← (18 ρ  a) / 'Population extinct'
  → E × a
  E:
    a ← ∨⌿ mat
    c ← a ⍳ 1
    h ← 2 + ρmat
    a ← h[2] − ((⌽a) ⍳ 1) + c
    b ← ∨/ mat
    c ← b ⍳ 1
    b ← h[1] − ((⌽b) ⍳ 1) + c
    mat ← (0, (b ρ 1), 0) ⍀ (0, (a ρ 1), 0)
        \mat[¯1 + c + ⍳b]
    h ← ρmat
    ⎕←'Generation ';Age;'. *'[1+(0,(h[1]ρ1)
        ,0)⍀(0,h[2]ρ1),0)\mat+1]
    Age ← Age + 1
  → 0 × ⍳(Gen < Age)
    h ← ¯1 ⊖ mat ⍝
    k ← 1 ⊖ mat ⍝
    h←(1⌽mat)+(1⌽k)+(¯1⌽k)+k+(1⌽h)+(¯1⌽h)+(
        ¯1⌽mat)+h ⍝ (*)
    mat ← (h = 3) ∨ mat ∧ 2 = h ⍝
    h ← 0 ρ mat ⍝
  → St
   ∇
```

```
PRIMES : (~R∊R∘.×R)/R←1↓⍳R
```

OK, Notation helps thinking. Why *ELSE* Should We Study Programming Languages?

- Improved background for choosing appropriate tools / languages
- Increased ability to learn “new” languages or design new ones
- Better understanding of the interchange between implementation and design
- Appreciation of the beauty of relevant material from **CS 61a/b/c CS170, Math 55**

Course Structure

- Course has theoretical and practical aspects
- Need both in programming languages!
- Occasional written assignments = theory
  - Class hand-in
- Programming assignments = practice
  - Electronic hand-in

Academic Honesty

- Re-using programs is an important engineering technique.
- BUT **Please** don't use work from uncited sources
- (The Easy and Correct Solution: cite ALL sources, including friends, TAs, staff, old projects, partners.)
- Please be sure that you understand what you hand in.
- (it is NOT OK to say "my partner did that and I don't know how it works")
- If you have questions about academic honesty, (yours or others) ask some faculty member.

Objectives of the Course

- This course is intended to be a learning experience, not an exercise in debugging.
- The homeworks and programs are primarily for you to learn about programming languages and compilers.
- Programming is not an endurance test. Delaying your project until the last minute and then staying up all night to complete it is not a good way to learn.
- Unfortunately, assessment is part of the deal. We will grade you on your exams (especially), but also on your homeworks, to keep you motivated. (Details on handout).

The Course Project

- A "big" project. Written in Common Lisp.
- … in several easy parts
- Why Lisp?
  - Typical JAVA/C++ project code size: 5,000 lines
  - Typical Lisp code size: 2,000 lines.
  - You are given 1000 lines in "skeletons" either way, so the ratio is about 4:1 in favor of Lisp
  - You all know Scheme, or at least you used to know it; CL is like Scheme on steroids.

You have been writing programs for years– what more is there to learn?

- How do Basic, Pascal, C, C++, Assembler, Java, Scheme differ? And WHY?
- What are the essential common threads in design (Variables? Arithmetic? Subroutines?)
- What are the essential common threads in implementation? Do you remember anything from CS61a language implementation? (Some of you did a project to partially implement Logo).

Overview: How are Languages Implemented?

- Two major strategies:
  - Interpreters (simple, general, faster setup)
  - Compilers (complex, popular, slower setup, faster at runtime)

- Interpreters run programs with only modest "digestion"; often easily portable to many hosts.

- Compilers do extensive digestion, some of it specific to particular machine architectures. Often not portable.
- We will study both strategies, as well as mixtures (e.g. Java VM, Lisp)

Common: Interpreters and Compilers both

- Read in “source code” text.
- Construct *some model* of the program.
- Provide error messages.
- Adhere to some “execution model” of the programming language.

- Important: the human programmer and the computer must “agree” on this model: what is the meaning of a program?

An orthogonal issue that is sometimes discussed: Is a language Batch or Interactive?

- Batch compilation systems dominate many production environments; Optimizes for fast runtime. Sacrifices debugging. (E.g. Fortran, C).
- Some languages are meant to be used interactively. E.g. Matlab, Lisp, Logo. They could be compiled or interpreted.

- Lisp implementations provides both
  - Interpreter for faster development/debugging
  - Type-checking compiler for finding more kinds of bugs, and faster running

There are Many Programming Languages

- Fortran, Algol, Lisp, COBOL, APL, BASIC, Smalltalk, B,C,C++, C#, Java, etc etc… we will talk about some of these next time in an historical context.
- Thought for today:
  - Good notation can help us think. PLs are notation:
    - Different approaches
    - Different applications
    - Significant similarities

Break...

- Handouts
- Demo of Lisp
- Questions?

Let's Get Started with the Course Material!

The Structure of a Typical Compiler

1. Lexical Analysis
2. Parsing
3. Semantic Analysis
4. Optimization
5. Code Generation

The first 3, at least, can be understood by analogy to how humans comprehend English.

Lexical Analysis (The "lexer" or "scanner")

- First step: recognize words.
  - Smallest unit above letters

This is a sentence.

- Note the
  - Capital "T" (start of sentence symbol)
  - Blank " " (word separator)
  - Period "." (end of sentence symbol)

More Lexical Analysis

- Human Lexical analysis is not so trivial. Consider:

  ist his ase nte nce

- Plus, programming languages are typically more cryptic than English:

  *p->f ++ = -.12345e-5

And More Lexical Analysis

- A Lexical analyzer divides program text into "words" or "tokens" not just at "white space".

  if x == y then z = 1; else z = 2;

- Units:

  if, x, ==, y, then, z, =, 1, ;, else, z, =, 2, ;

- Some tokens are <u>operators</u>, some <u>identifiers</u>, some are <u>numbers</u> and some are "<u>keywords</u>".

Parsing in a nutshell (we will spend several weeks elaborating on this, though)

- Once words are understood, the next step is to understand "sentence structure".
- In fact, the term "sentence" can be used technically as in

"Parsing a Java program is determining if a text is a sentence in the Java grammar."

- Parsing = Diagramming Sentences by grammar rules
  - The diagram is a tree

Diagramming an English Sentence