Lecture 20: Little Languages
Over-simplified history of programming languages

• 1940's machine language
• 1950's assembly language
• 1960's high-level languages: ♦ scripting languages:
  Algol, Fortran, Cobol, Basic  ♦ Snobol
• 1970's systems programming: C
• 1980's object-oriented: C++
• 1990's strongly-hyped: Java
• 2000's lookalike languages: C#
• 2010's retry: Go, Rust, Swift

♦ shell ♦ Awk ♦ Perl, Python, PHP, … ♦ Javascript ♦ Dart, Typescript
Domain-specific languages

• also called application specific languages, little languages

• narrow domain of applicability
• not necessarily programmable or Turing-complete
  – often declarative, not imperative
• often small enough that you could build one yourself

• examples:
  – regular expressions
  – parser and lexer generators: YACC, LEX, ANTLR
  – shell, Awk
  – markup languages: XML, HTML, Troff, (La)TeX, Markdown
  – data format/exchange languages: YAML, JSON, ASN.1
  – database access: SQL
  – statistics: R
  – mathematical optimization: AMPL
  – ...
Example: Markup / document preparation languages

• illustrates topics of 333 in a different setting
  – tools
  – language design (good and bad); notation
  – evolution of software systems; maintenance
  – personal interest, research area for 10-20 years, heavy use in books

• examples:
  – roff and related early formatters
  – nroff (Unix man command still uses it)
  – troff
  – Tex / Latex
  – HTML, Markdown, etc.
Unix document preparation: *roff

- text interspersed with formatting commands on separate lines
  
  .sp 2
  .in 5
  
  This is a paragraph ...

- originally just ASCII output, fixed layout, single column
- nroff: macros, a event mechanism for page layout (Turing complete)
- troff: version of nroff for phototypesetters
  – adds features for size, font, precise positioning, bigger character sets
  – originally by Joe Ossanna (~1972); inherited by BWK ~1977

- phototypesetter produces output on photographic paper or film
- first high-quality output device at a reasonable price (~$15K)
  – predates laser printers by 5-10 years
  – predates Postscript (1982) by 10 years, PDF (1993) by 21 years
  – klunky, slow, messy (chemicals!), expensive media

- complex program, complex language
  – language reflects many of the weirdnesses of first typesetter
  – macro packages make it usable by mortals for standard tasks

- troff + phototypesetter enables book-quality output
  – ..., K&R, TPOP, Go, …
Extension to complex specialized technical material

- mathematics
  - called “penalty copy” in the printing industry
- tables
- drawings
- graphs
- references
- indexes
- etc.

- at the time, done by hand composition
  - not much better than medieval technology

- Bell Labs authors writing papers and books with all of these
- being done by manual typewriters
- how can production be mechanized?
EQN: a language for typesetting mathematics

• BWK, with Lorinda Cherry ~1974

• idea: a language that matches the way mathematics is spoken aloud

• translate that into troff commands
  – since the language is so orthogonal, it wouldn’t fit directly
  – and there isn’t room anyway, since program has to be less than 65KB
  – troff is powerful enough

• use a pipeline: `eqn | troff`

• math mode in TEX (1978) was inspired by EQN
EQN examples

\[ x^2 + y^2 = z^2 \]

\[ f(t) = 2\pi \int \sin (\omega t) \, dt \]

\[ \lim_{x \to \pi/2} (\tan x) = \infty \]

\[ x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \]
**EQN implementation**

- based on a YACC grammar
  - first use of YACC outside mainstream compilers

- grammar is simple
  - box model
  - just combine boxes in various ways:
    - concatenate, above/below, sub and superscript, sqrt, ...

```
eqn: box | eqn box
box: text | { eqn } | box over box | sqrt box
   | box sub box | box sup box | box from box to box | ...
```

- YACC makes experimental language design easy
Pic: a language for pictures (line drawings)

- new typesetter has more capabilities  (costs more too: $50K in 1977)
- can we use troff to do line drawings?

- answer: invent another language, again a preprocessor
  - add simple line-drawing primitives to troff: line, arc, spline

- advantages of text descriptions of pictures
  - systematic changes are easy, always have correct dimensions,
  - Pic has loops, conditionals, etc., for repetitive structures
    Turing complete!

- implemented with YACC and LEX
  - makes it easy to experiment with syntax
  - human engineering:
    free-form English-like syntax
    implicit positioning: little need for arithmetic on coordinates
Pic examples

```
.PS
arrow "input" above
box "process"
arrow "output" above
.PE
```
Pic examples

."PS
V: arrow from 0,-1 to 0,1;  " voltage" ljust at V.end
L: arrow from 0,0 to 4,0;  " time" ljust at L.end
for i = 1 to 399 do X
    j = i+1
    line from (L + i/100, sin(i/10) / 3 + sin(i/20) / 2 + sin(i/30) / 4) to (L + j/100, sin(j/10) / 3 + sin(j/20) / 2 + sin(j/30) / 4)
X
."PE
Grap: a language for drawing graphs

- line drawings, not “charts” in the Excel sense
- with Jon Bentley, ~1984

- a Pic preprocessor: `grap | pic | troff`

```
.G1
0 0
1 1
2 4
3 9
4 16
5 25
.G2
```
The Go Programming Language experience

- started with Markdown
  - very good for simple documents
- doesn't scale to books
  - too many special cases if material is complicated (e.g., fonts, layout)
  - very slow
- Alan Donovan wrote a version in Go
  - better, but still too many special cases
- LaTeX?
  - it's complicated, inflexible and uncontrollable
- convert book text to XML, process by a Go program
  - about 20 tags, with attributes
  - a nuisance to type, but many fewer special cases
  - generates HTML for proofing and ultimately ebooks
  - generates Troff for paper version
  - still lots of special-purpose shell scripts, e.g., indexing, special chars
AMPL: A big DSL that got bigger

- a language and system for
  - describing optimization problems in a uniform, natural way
  - compiling descriptions into form needed by solver programs
  - controlling execution of solvers
  - displaying results in problem terms

Robert Fourer
David Gay
Brian Kernighan
Cost minimization: a diet model

- Find a minimum-cost mix of TV dinners that satisfies requirements on the minimum and maximum amounts of certain nutrients.
- Given sets and parameters:
  - $F$, a set of foods
  - $N$, a set of nutrients
  - $a_{ij} =$ amount of nutrient $i$ in a package of food $j$
  - $c_j =$ cost of package of food $j$, for each $j \in F$
  - $f_j^-$ = minimum packages of food $j$, for each $j \in F$
  - $f_j^+$ = maximum packages of food $j$, for each $j \in F$
  - $n_i^-$ = minimum amount of nutrient $i$, for each $i \in N$
  - $n_i^+$ = maximum amount of nutrient $i$, for each $i \in N$
- Define decision variables:
  - $X_j =$ packages of food $j$ to buy, for each $j \in F$
- Minimize objective:
  - $\sum_{j \in F} c_j X_j$
- Subject to constraints:
  - $n_i^- \leq \sum_{j \in F} a_{ij} X_j \leq n_i^+$, for each $i \in N$
  - $f_j^- \leq X_j \leq f_j^+$, for each $j \in F$
AMPL version of the diet model

set FOOD;
set NUTR;

param amt {NUTR,FOOD} >= 0;
param cost {FOOD} > 0;
param f_min {FOOD} >= 0;
param f_max {j in FOOD} >= f_min[j];
param n_min {NUTR} >= 0;
param n_max {i in NUTR} >= n_min[i];

var Buy {j in FOOD} >= f_min[j], <= f_max[j];

minimize total_cost:  sum {j in FOOD} cost[j] * Buy[j];

subject to diet {i in NUTR}:
   n_min[i] <= sum {j in FOOD} amt[i,j] * Buy[j] <= n_max[i];
Build optimization into your large-scale applications — quickly and reliably — using AMPL’s powerful yet intuitive algebraic modeling system.

**AMPL FOR BUSINESS**  
Streamlined optimization development in business applications of all kinds.  
Read More

**AMPL FOR TEACHING**  
Free AMPL and solvers. Full-featured, time-limited. Easy to install & distribute.  
Read More

**AMPL FOR RESEARCH**  
Optimization modeling for engineering, science, economics, management.  
Read More

**SOLVERS**  
Buy from us >>  
CPLEX • Gurobi • Knitro • Xpress  
CONOPT • LINDO • MINOS • SNOPT — BARON • LGO  
Open-source optimizers >>  
Full solver list >>  

**WHY AMPL?**  
The AMPL system supports the entire optimization modeling lifecycle — formulation, testing, deployment, and maintenance — in an integrated way promotes rapid development and reliable results. Using a high-level algebraic representation that describes optimization models in the same ways that people think about them, AMPL can provide the head start you need to successfully implement large-scale optimization projects.

AMPL integrates a modeling language for describing optimization data, variables, objectives, and constraints; a command language for debugging models and analyzing results; and a scripting language for manipulating data and implementing optimization strategies. All use...
Why languages succeed

• solve real problems effectively

• culturally compatible and familiar
  – familiar syntax helps (e.g., C-like)
  – easy to get started with
  – portable to new environments

• environmentally compatible
  – don’t have to buy into an entire new environment to use it
  – e.g., can use standard tools and link to existing libraries
  – open source, not proprietary

• weak competition

• good luck
Why languages fail to thrive

• niche or domain disappears

• poor engineering
  – too big, too complicated, too slow, too late
  – incompatible with environments

• poor philosophical choices
  – ideology over functionality
  – single programming paradigm
  – too "mathematical"
  – too different, too incompatible