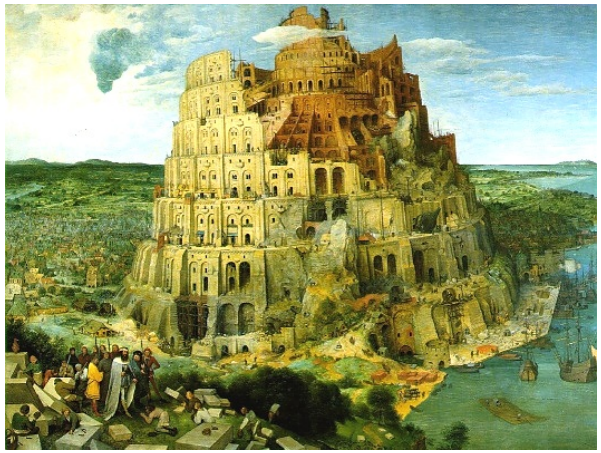
Programming Languages and Translators

Stephen A. Edwards

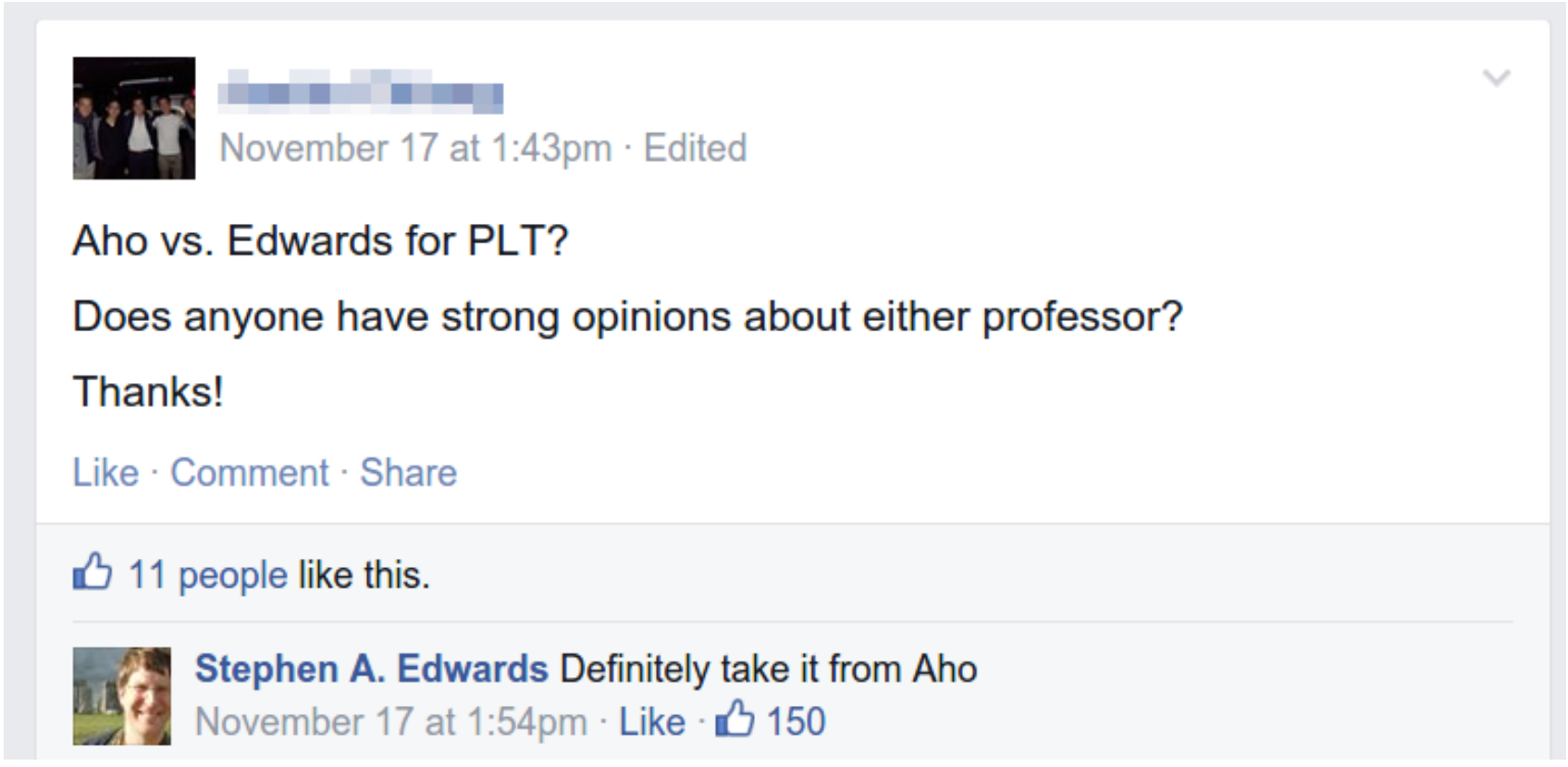
Columbia University

Summer 2016



Pieter Bruegel, *The Tower of Babel*, 1563

Facebook on 4115



Sadly, Aho has retired.

Instructor

Prof. Stephen A. Edwards

sedwards@cs.columbia.edu

http://www.cs.columbia.edu/~sedwards/

462 Computer Science Building

Email me for appointments

Culpa on 4115

Edwards is the snarkiest, most sarcastic, immature professor you will meet in the CS department. He tells some really great nerdy jokes and his Facebook wall is hilarious since he belittles all his students publicly on it, but I don't recommend taking his class. Don't ever email him with an excuse or stupid question since he will publicly shame you (name removed though) on Facebook.

Objectives

Theory

- Principles of modern programming languages
- Fundamentals of compilers: parsing, type checking, code generation
- Models of computation

Practice: Semester-long Team Project

- Design and implement your own language and compiler
- Code it in the OCaml functional language
- Manage the project and your teammates; communicate

Quasi-required Text

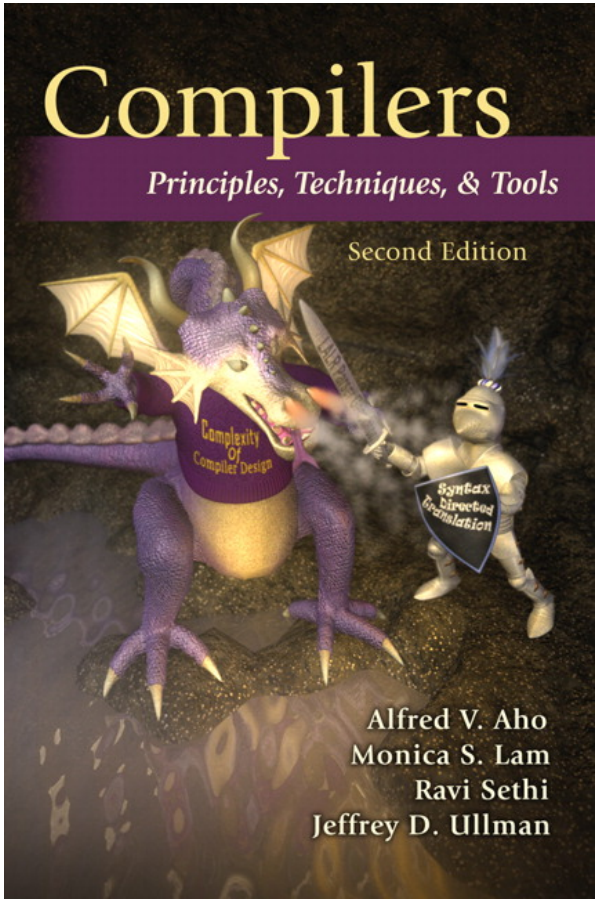
Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman.

Compilers: Principles, Techniques, and Tools.

Addison-Wesley, 2006. Second Edition.

Bug Al about all bugs.

You can get away with the first edition.



Schedule

Lectures: Mondays and Wednesdays, 5:30 – 8:40 PM

July 6 – August 10

Final: Wednesday, August 10

Presentations: August 11

Final project reports: August 11

You can present before August 11 if you want to travel early. All group members must present.

Assignments and Grading

50% Programming Project

40% Final

10% Two individual homework assignments

Project is most important, but most students do well on it.
Grades for tests often vary more.

Prerequisites

COMS W3157 Advanced Programming

- How to work on a large software system in a team
- Makefiles, version control, test suites
- Testing will be as important as development

COMS W3261 Computer Science Theory

- Regular languages and expressions
- Context-free grammars
- Finite automata (NFAs and DFAs)

Collaboration

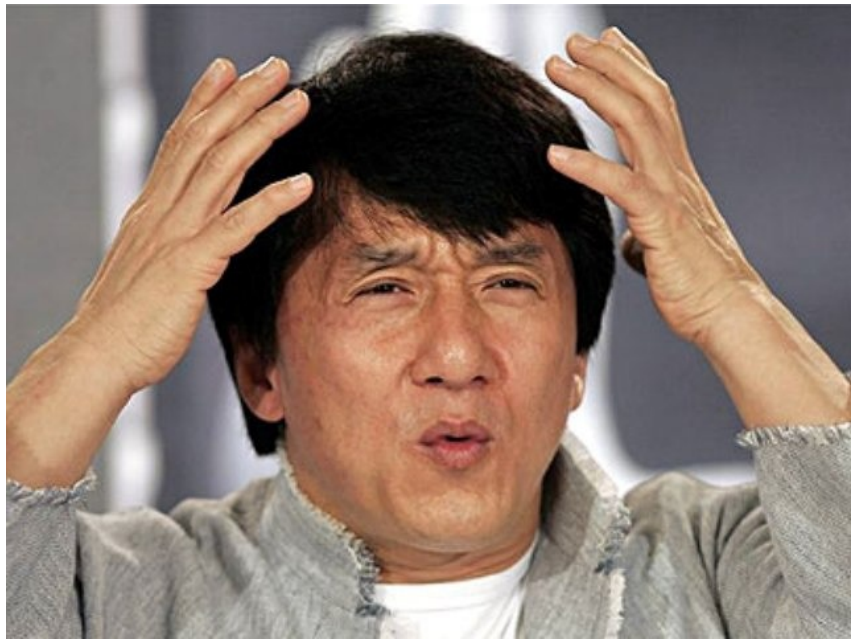
Collaborate with your team on the project.

Do your homework by yourself.

Tests: Will be closed book with a one-page "cheat sheet" of your own devising.

Don't be a cheater (e.g., copy from each other):
If you're dumb enough to cheat,
I'm smart enough to catch you.

Every term I've caught cheaters and sent them to the dean.
Please try to break my streak.



The Project

The Project

Design and implement your own little language.

Six deliverables:

1. A proposal describing your language
2. A language reference manual defining it formally
3. An intermediate milestone: compiling “Hello World.”
4. A compiler for it, running sample programs
5. A final project report
6. A final project presentation

Teams

Immediately start forming four-person teams

Each team will develop its own language

Assign each team member a specific role

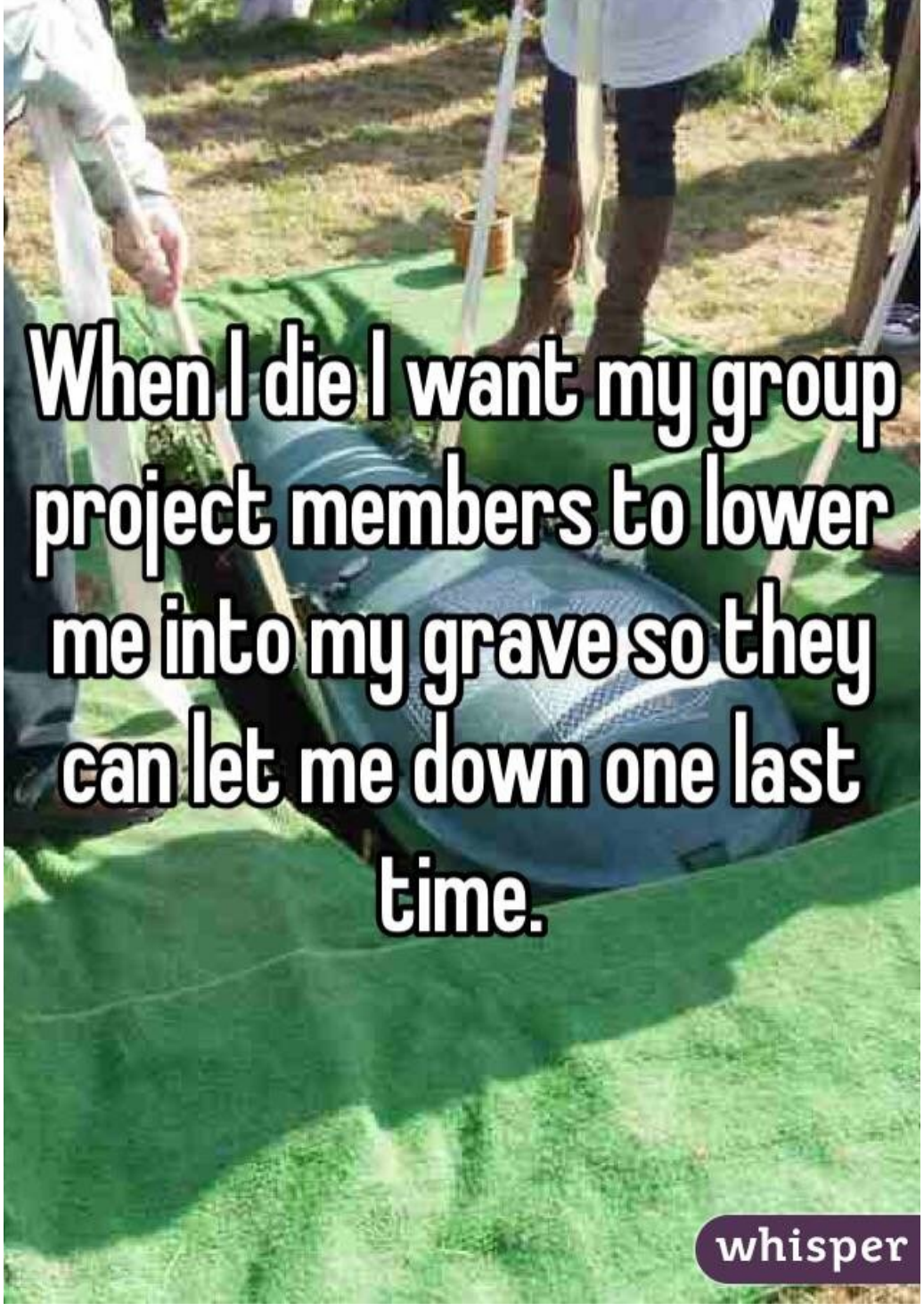
Role	Responsibilities
Manager	Timely completion of deliverables
Language Guru	Language design
System Architect	Compiler architecture, development environment
Tester	Test plan, test suites

Culpa Suggestion

START EARLY, and really be selective in picking your team. A bad team will ruin the semester for you.

EVERY GROUP PROJECT

IN SCHOOL YOU HAVE EVER DONE


When I die I want my group project members to lower me into my grave so they can let me down one last time.
whisper

First Three Tasks

1. Decide who you will work with
   You'll be stuck with them for the term; choose wisely.
2. Assign a role to each member
   Languages come out better from dictatorships, not democracies.
3. Select a weekly meeting time
   Harder than you might think.

Project Proposal

Describe the language that you plan to implement.

Explain what sorts of programs are meant to be written in your language

Explain the parts of your language and what they do

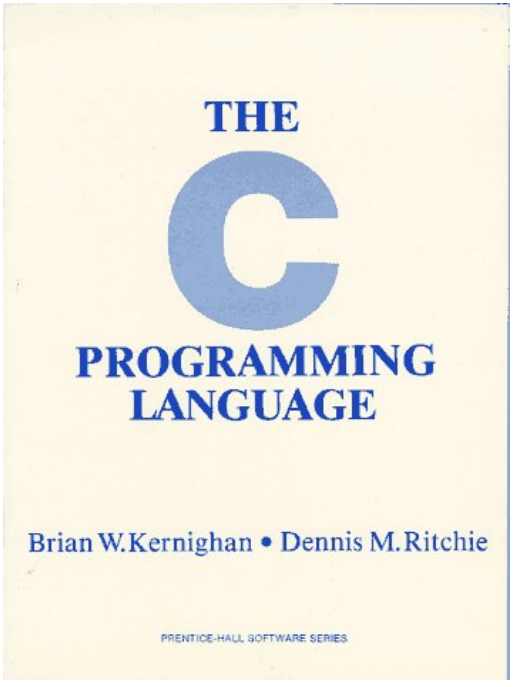
Include the source code for an interesting program in your language

2–4 pages

Language Reference Manual

A careful definition of the syntax and semantics of your language.

Follow the style of the C language reference manual (Appendix A of Kernighan and Ritchie, *The C Programming Langauge*; see the class website).



Final Report Sections

Section	Author
Introduction	Team
Tutorial	Team
Reference Manual	Team
Project Plan	Manager
Language Evolution	Language Guru
Translator Architecture	System Architect
Test plan and scripts	Tester
Conclusions	Team
Full Code Listing	Team

Project Due Dates

Proposal	July 11 soon
Language Reference Manual	July 20
Hello World Demo	August 1
Final Report	August 11

Design a language?

A domain-specific language: awk or PHP, not Java or C++.

Examples from earlier terms:

Matlab-like array manipulation language

Geometric figure drawing language

Screenplay animation language

Escher-like pattern generator

Music manipulation language

Mathematical function manipulator

Simple scripting language (à lá Tcl)

Three Common Mistakes to Avoid

Configuration File Syndrome

- Must be able to express *algorithms*, not just data
- E.g., a program like "a bird and a turtle and a pond and grass and a rock," is just data, not an algorithm

Standard Library Syndrome

- Good languages express lots by a combining few things
- Write a standard library in your language
- Aim for Legos, not Microsoft Word

Java-to-Java Translator Syndrome

- A compiler mostly adds implementation details to code
- Your compiler's output should not look like its input
- Try your best not to re-invent Java

What I'm Looking For

Your language must be able to express different algorithms

- Avoid Configuration File Syndrome. Most languages should be able to express, e.g., the GCD algorithm.

Your language should consist of pieces that can mix freely

- Avoid Standard Library Syndrome. For anything you provide in the language, ask yourself whether you can express it using other primitives in your language.

Your compiler must lower the level of abstraction

- Don't write a Java-to-Java translator. Make sure your compiler adds details to the output such as registers, evaluation order of expressions, stack management instructions, etc.

What’s in a Language?

Components of a language: Syntax

How characters combine to form words, sentences, paragraphs.

The quick brown fox jumps over the lazy dog.

is syntactically correct English, but isn't a Java program.

```
class Foo {
  public int j;
  public int foo(int k) { return j + k; }
}
```

is syntactically correct Java, but isn't C.

Specifying Syntax

Usually done with a context-free grammar.

Typical syntax for algebraic expressions:

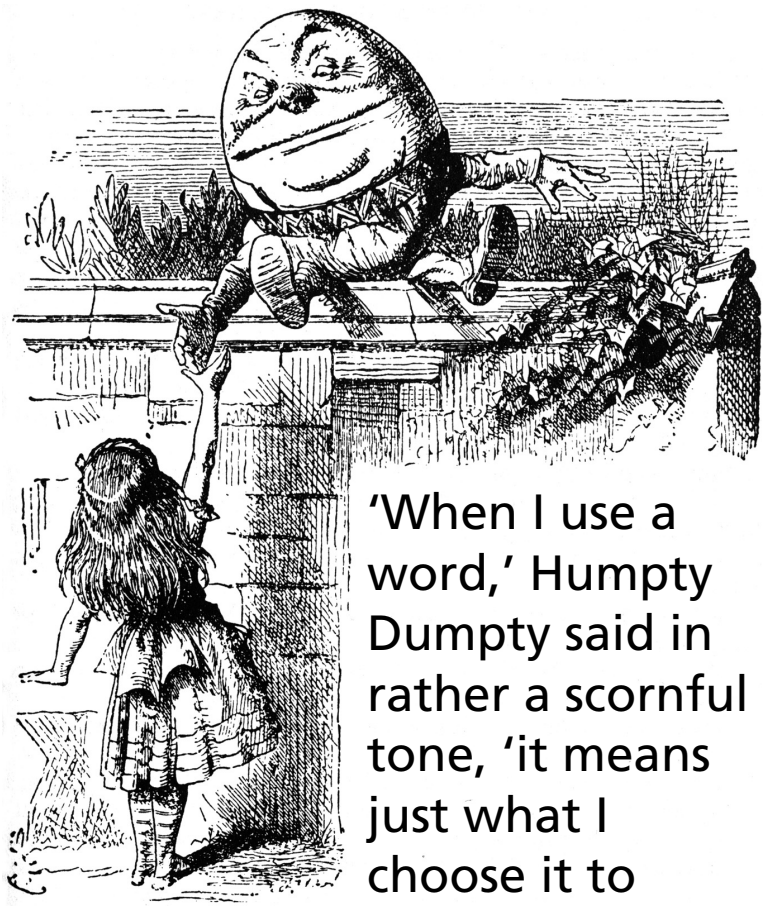
$$
\begin{array}{lcl}
\textit{expr} & \rightarrow & \textit{expr} + \textit{expr} \\
& | & \textit{expr} - \textit{expr} \\
& | & \textit{expr} * \textit{expr} \\
& | & \textit{expr} \,/\, \textit{expr} \\
& | & \textbf{digit} \\
& | & (\textit{expr})
\end{array}
$$

Components of a language: Semantics

What a well-formed program "means."

The semantics of C says this computes the nth Fibonacci number.

```
int fib(int n)
{
  int a = 0, b = 1;
  int i;
  for (i = 1 ; i < n ; i++) {
    int c = a + b;
    a = b;
    b = c;
  }
  return b;
}
```



'When I use a word,' Humpty Dumpty said in rather a scornful tone, 'it means just what I choose it to mean—neither more nor less.'

Source: Lewis Carroll, *Through the Looking Glass*, 1872.

Semantics

Something may be syntactically correct but semantically nonsensical

The rock jumped through the hairy planet.

Or ambiguous

The chickens are ready to eat.

Semantics

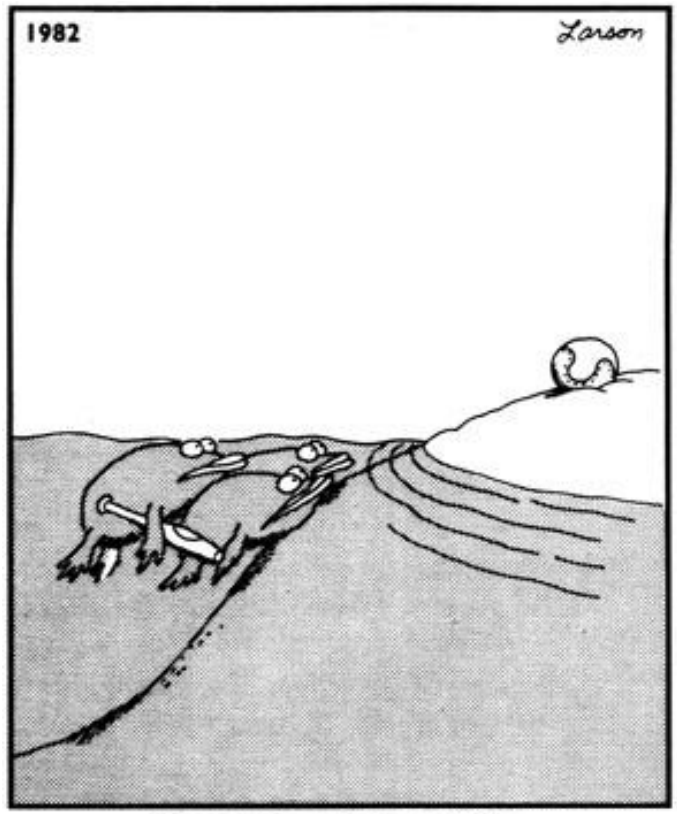
Nonsensical in Java:

```
class Foo {
  int bar(int x) { return Foo; }
}
```

Ambiguous in Java:

```
class Bar {
  public float foo() { return 0; }
  public int foo() { return 0; }
}
```

Great Moments in Evolution



Great moments in evolution

Assembly Language

Before: numbers

```
55
89E5
8B4508
8B550C
39D0
740D
39D0
7E08
29D0
39D0
75F6
C9
C3
29C2
EBF6
```

After: Symbols

```
gcd: pushl %ebp
     movl  %esp, %ebp
     movl  8(%ebp), %eax
     movl  12(%ebp), %edx
     cmpl  %edx, %eax
     je    .L9
.L7: cmpl  %edx, %eax
     jle   .L5
     subl  %edx, %eax
.L2: cmpl  %edx, %eax
     jne   .L7
.L9: leave
     ret
.L5: subl  %eax, %edx
     jmp   .L2
```

FORTRAN

Before

```
gcd: pushl %ebp
     movl  %esp, %ebp
     movl  8(%ebp), %eax
     movl  12(%ebp), %edx
     cmpl  %edx, %eax
     je    .L9
.L7: cmpl  %edx, %eax
     jle   .L5
     subl  %edx, %eax
.L2: cmpl  %edx, %eax
     jne   .L7
.L9: leave
     ret
.L5: subl  %eax, %edx
     jmp   .L2
```

After: Expressions, control-flow

```
 10   if (a .EQ. b) goto 20
      if (a .LT. b) then
         a = a - b
      else
         b = b - a
      endif
      goto 10
 20   end
```

99 Bottles of Beer in FORTRAN

```
      program ninetyninebottles
      integer bottles
      bottles = 99
 1    format (I2, A)
 2    format (A)
 3    format (I2, A, /)
 4    format (A, /)
 10   write (*,1) bottles, ' bottles of beer on the wall,'
      write (*,1) bottles, ' bottles of beer.'
      write (*,2) 'Take one down, pass it around...'
      if (bottles - 1 .gt. 1) then
         write (*,3) bottles - 1, ' bottles of beer on the wall.'
      else
         write (*,3) bottles - 1, ' bottle of beer on the wall.'
      end if
      bottles = bottles - 1
      if (bottles - 1) 30, 20, 10
*     Last verse
 20   write (*,1) bottles, ' bottle of beer on the wall,'
      write (*,1) bottles, ' bottle of beer.'
      write (*,2) 'Take one down, pass it around...'
      write (*,4) 'No bottles of beer on the wall.'
 30   stop
      end
```

99 Bottles of Beer in FORTRAN

```
      program ninetynineboti
      integer bottles
      bottles = 99
 1    format (I2, A)
 2    format (A)
 3    format (I2, A, /)
 4    format (A, /)
 10   write (*,1) bottles, '
      write (*,1) bottles, '
      write (*,2) 'Take one
      if (bottles - 1 .gt. 1
         write (*,3) bottles
      else
         write (*,3) bottles
      end if
      bottles = bottles - 1
      if (bottles - 1) 30, 2
*     Last verse
 20   write (*,1) bottles, '
      write (*,1) bottles, '
      write (*,2) 'Take one
      write (*,4) 'No bottle
 30   stop
      end
```

Backus, IBM, 1956

Imperative language for science and engineering

First compiled language

Fixed format lines (for punch cards)

Arithmetic expressions, If, Do, and Goto statements

Scalar (number) and array types

Limited string support

Still common in high-performance computing

Inspired most modern languages, especially BASIC

COBOL

Added type declarations, record types, file manipulation

```
data division.
file section.
*   describe the input file
fd  employee-file-in
            label records standard
            block contains 5 records
            record contains 31 characters
            data record is employee-record-in.
01  employee-record-in.
    02  employee-name-in  pic x(20).
    02  employee-rate-in  pic 9(3)v99.
    02  employee-hours-in pic 9(3)v99.
    02  line-feed-in      pic x(1).
```

Grace Hopper et al.

LISP, Scheme, Common LISP

Functional, high-level languages

```
(defun gnome-doc-insert ()
  "Add a documentation header to the current function.
Only C/C++ function types are properly supported currently."
  (interactive)
  (let (c-insert-here (point))
    (save-excursion
      (beginning-of-defun)
      (let (c-arglist
            c-funcname
            (c-point (point))
            c-comment-point
            c-isvoid
            c-doinsert)
        (search-backward "(")
        (forward-line -2)
        (while (or (looking-at "^$")
                   (looking-at "^ *}")
                   (looking-at "^ \\*")
                   (looking-at "^#"))
          (forward-line 1))
```

99 Bottles of Beer in LISP

```
(defun bottles-of-bier (n)
  (case n
   (0
    '(No more bottles of beer on the wall no more bottles of beer.
        Go to the store and buy some more 99 bottles of beer on the w
   (1
    `(1 bottle of beer on the wall 1 bottle of beer.
       Take one down and pass it around no more bottles of beer on th
       ,@(bottles-of-bier 0)))
   (2
    `(2 bottles of beer on the wall 2 bottles of beer.
      Take one down and pass it around 1 bottle of beer on the wall.
       ,@(bottles-of-bier 1)))
   (t
    `(,n bottles of beer on the wall ,n bottles of beer.
        Take one down and pass it around
        ,(1- n) bottles of beer on the wall.
        ,@(bottles-of-bier (1- n))))))
```

jimka, http://www.99-bottles-of-beer.net/language-lisp-1465.html

99 Bottles of Beer in LISP

```
(defun bottles-of-bier (n)
  (case n
   (0
    '(No more bottles of bee
        Go to the store and
   (1
    `(1 bottle of beer on th
       Take one down and pa
       ,@(bottles-of-bier 0
   (2
    `(2 bottles of beer on t
      Take one down and pas
      ,@(bottles-of-bier 1)
   (t
    `(,n bottles of beer on
        Take one down and p
        ,(1- n) bottles of
        ,@(bottles-of-bier
```

- McCarthy, MIT, 1958
- Functional: recursive, list-focused functions
- Semantics from Church's Lambda Calculus
- Simple, heavily parenthesized S-expression syntax
- Dynamically typed
- Automatic garbage collection
- Originally for AI applications
- Dialects: Scheme and Common Lisp

jimka, `http://www.99-bottles-of-beer.net/language-lisp-1465.html`

APL

Powerful operators, interactive language, custom character set

```
[0]    Z←GAUSSRAND N;B;F;M;P;Q;R
[1]    ⍝Returns ⍵ random numbers having a Gaussian normal distribution
[2]    ⍝ (with mean 0 and variance 1)  Uses the Box-Muller method.
[3]    ⍝ See Numerical Recipes in C, pg. 289.
[4]    ⍝
[5]    Z←⍳0
[6]    M←¯1+2*31          ⍝ largest integer
[7]   L1:Q←N-⍴Z           ⍝ how many more we need
[8]    →(Q≤0)/L2          ⍝ quit if none
[9]    Q←⌈1.3×Q÷2         ⍝ approx num points needed
[10]   P←¯1+(2÷M-1)×¯1+?(Q,2)⍴M  ⍝ random points in -1 to 1 square
[11]   R←+/P×P            ⍝ distance from origin squared
[12]   B←(R≠0)∧R<1
[13]   R←B/R ⋄ P←B⌿P      ⍝ points within unit circle
[14]   F←(¯2×(⊕R)÷R)*.5
[15]   Z←Z,,P×F,[1.5]F
[16]   →L1
[17]  L2:Z←N↑Z
[18]   ⍝ ArchDate: 12/16/1997 16:20:23.170
```

"Emoticons for Mathematicians"

Source: Jim Weigang, http://www.chilton.com/~jimw/gsrand.html

At right: Datamedia APL Keyboard

99 Bottles of Beer in APL

```
⍝ APL (A Programming Language)

⍝ Program written by JT. Taylor, www.jttaylor.net

T1←98↑[1]⍬⍴1 99⍴⍳99

T4←⍬⍴1 98⍴⍳98

T1,(98 30⍴' BOTTLES OF BEER ON THE WALL, '),T1,
(98 47⍴'BOTTLES OF BEER, TAKE ONE DOWN, PASS IT
AROUND,'),T4,(98 28⍴'BOTTLES OF BEER ON THE
WALL ,')

'1 BOTTLE OF BEER ON THE WALL, 1 BOTTLE OF BEER,
TAKE IT DOWN, PASS IT AROUND, NO BOTTLES OF BEER
ON THE WALL.'
```

http://www.99-bottles-of-beer.net/language-apl-715.html

99 Bottles of Beer in APL

```
⍝ APL (A Programming Lan

⍝ Program written by JT.

T1←98↑[1]⍬⍳1 99⍴⍳99

T4←⍬⍳1 98⍴⍳98

T1,(98 30⍴' BOTTLES OF B
(98 47⍴'BOTTLES OF BEER,
AROUND,'),T4,(98 28⍴'BOT
WALL ,')

'1 BOTTLE OF BEER ON THE
TAKE IT DOWN, PASS IT AR
ON THE WALL.'
```

Iverson, IBM, 1960

Imperative, matrix-centric

E.g., perform an operation on each element of a vector

Uses own specialized character set

Concise, effectively cryptic

Primarily symbols instead of words

Dynamically typed

Odd left-to-right evaluation policy

Useful for statistics, other matrix-oriented applications

http://www.99-bottles-of-beer.net/language-apl-715.html

Algol, Pascal, Clu, Modula, Ada

Imperative, block-structured language, formal syntax definition, structured programming

```
PROC insert = (INT e, REF TREE t)VOID:
   # NB inserts in t as a side effect #
   IF TREE(t) IS NIL THEN
     t := HEAP NODE := (e, TREE(NIL), TREE(NIL))
   ELIF e < e OF t THEN insert(e, l OF t)
   ELIF e > e OF t THEN insert(e, r OF t)
   FI;

 PROC trav = (INT switch, TREE t, SCANNER continue,
              alternative)VOID:
   # traverse the root node and right sub-tree of t  only. #
   IF t IS NIL THEN continue(switch, alternative)
   ELIF e OF t  <=  switch THEN
         print(e OF t);
         traverse( switch, r OF t, continue, alternative)
   ELSE  # e OF t > switch #
         PROC defer = (INT sw, SCANNER alt)VOID:
               trav(sw, t, continue, alt);
         alternative(e OF t, defer)
   FI;
```

Algol-68, source http://www.csse.monash.edu.au/~lloyd/tildeProgLang/Algol68/treemerge.a68

SNOBOL, Icon

String-processing languages

```
   LETTER  =  'ABCDEFGHIJKLMNOPQRSTUVWXYZ$#@'
   SP.CH   =  "+-,=.*()'/& "
   SCOTA  =  SP.CH
   SCOTA   '&'  =
   Q  =  "'"
   QLIT  =  Q  FENCE  BREAK(Q)  Q
   ELEM  =  QLIT | 'L' Q | ANY(SCOTA) | BREAK(SCOTA) | REM
   F3  =  ARBNO(ELEM FENCE)
   B  =  (SPAN(' ') | RPOS(0))  FENCE
   F1  =  BREAK(' ')  |  REM
   F2  =  F1
   CAOP  =  ('LCL'  |  'SET')  ANY('ABC')  |
+  'AIF'  |  'AGO'  |  'ACTR'  |  'ANOP'
   ATTR  =  ANY('TLSIKN')
   ELEMC  =  '(' FENCE *F3C ')'  |  ATTR Q  |  ELEM
   F3C    =  ARBNO(ELEMC  FENCE)
   ASM360  =  F1 . NAME  B
+  ( CAOP . OPERATION  B  F3C . OPERAND  |
+  F2 . OPERATION    B  F3 . OPERAND)
+  B    REM . COMMENT
```

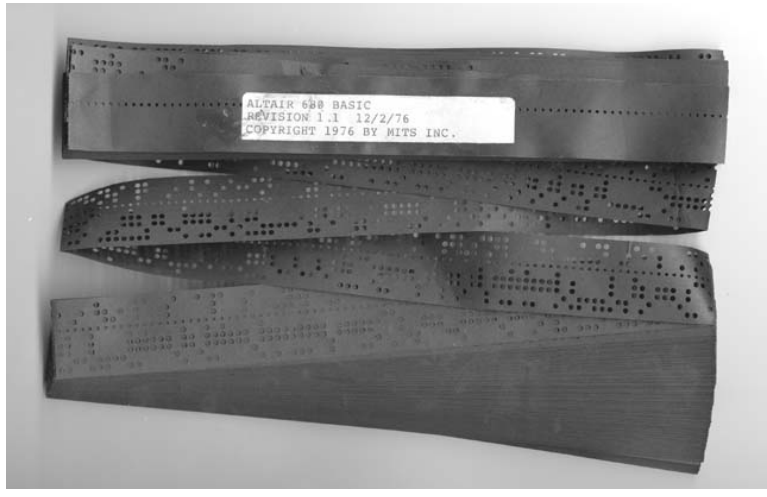
SNOBOL: Parse IBM 360 assembly. From Gimpel's book, http://www.snobol4.org/

BASIC

Programming for the masses

```
10 PRINT "GUESS A NUMBER BETWEEN ONE AND TEN"
20 INPUT A$
30 IF A$ <> "5" THEN GOTO 60
40 PRINT "GOOD JOB, YOU GUESSED IT"
50 GOTO 100
60 PRINT "YOU ARE WRONG. TRY AGAIN"
70 GOTO 10
100 END
```

Invented at Dartmouth by John George Kemeny and Thomas Eugene Kurtz. Started the whole Bill Gates/ Microsoft thing.



Simula, Smalltalk, C++, Java, C#

The object-oriented philosophy

```
class Shape(x, y); integer x; integer y;
virtual: procedure draw;
begin
   comment - get the x & y coordinates -;
   integer procedure getX;
      getX := x;
   integer procedure getY;
      getY := y;

   comment - set the x & y coordinates -;
   integer procedure setX(newx); integer newx;
      x := newx;
   integer procedure setY(newy); integer newy;
      y := newy;
end Shape;
```

99 Bottles of Beer in Java

```
class Bottles {
  public static void main(String args[]) {
    String s = "s";
    for (int beers=99; beers>-1;) {
      System.out.print(beers+" bottle"+s+" of beer on the wall, ");
      System.out.println(beers + " bottle" + s + " of beer, ");
      if (beers==0) {
        System.out.print("Go to the store, buy some more, ");
        System.out.println("99 bottles of beer on the wall.\n");
        System.exit(0);
      } else
        System.out.print("Take one down, pass it around, ");
      s = (--beers == 1)?"":"s";
      System.out.println(beers+" bottle"+s+" of beer on the wall.\n");
    }
  }
}
```

Sean Russell,
http://www.99-bottles-of-beer.net/language-java-4.html

99 Bottles of Beer in Java

```
class Bottles {
  public static void main(St
    String s = "s";
    for (int beers=99; beers
      System.out.print(beers
      System.out.println(bee
      if (beers==0) {
        System.out.print("Go
        System.out.println("
        System.exit(0);
      } else
        System.out.print("Ta
      s = (--beers == 1)?"":
      System.out.println(bee
    }
  }
}
```

- Gosling et al., Sun, 1991
- Imperative, object-oriented, threaded
- Based on C++, C, Algol, etc.
- Statically typed
- Automatic garbage collection
- Architecturally neutral
- Defined on a virtual machine (Java Bytecode)

Sean Russell,
`http://www.99-bottles-of-beer.net/language-java-4.html`

C

Efficiency for systems programming

```
int gcd(int a, int b)
{
  while (a != b) {
    if (a > b) a -= b;
    else b -= a;
  }
  return a;
}
```

99 Bottles of Beer in C

```c
#define MAXBEER 99
void chug(int beers);

int main()
{
  int beers;
  for(beers = MAXBEER; beers; chug(beers--)) ;
  puts("\nTime to buy more beer!\n");
  return 0;
}

void chug(int beers)
{
  char howmany[8], *s;
  s = beers != 1 ? "s" : "";
  printf("%d bottle%s of beer on the wall,\n", beers, s);
  printf("%d bottle%s of beeeeer . . . ,\n", beers, s);
  printf("Take one down, pass it around,\n");
  if (--beers) sprintf(howmany, "%d", beers);
  else strcpy(howmany, "No more");
  s = beers != 1 ? "s" : "";
  printf("%s bottle%s of beer on the wall.\n", howmany, s);
}
```

Bill Wein, http://www.99-bottles-of-beer.net/language-c-116.html

99 Bottles of Beer in C

```
#define MAXBEER 99
void chug(int beers);

int main()
{
  int beers;
  for(beers = MAXBEER; beers
  puts("\nTime to buy more b
  return 0;
}

void chug(int beers)
{
  char howmany[8], *s;
  s = beers != 1 ? "s" : "";
  printf("%d bottle%s of bee
  printf("%d bottle%s of bee
  printf("Take one down, pas
  if (--beers) sprintf(howma
  else strcpy(howmany, "No m
  s = beers != 1 ? "s" : "";
  printf("%s bottle%s of bee
}
```

Dennis Ritchie, Bell Labs, 1969

Procedural, imperative

Based on Algol, BCPL

Statically typed; liberal conversion policies

Harmonizes with processor architecture

For systems programming: unsafe by design

Remains language of choice for operating systems

Bill Wein, `http://www.99-bottles-of-beer.net/language-c-116.html`

ML, Miranda, Haskell

Functional languages with types and syntax

```
structure RevStack = struct
  type  'a stack =  'a list
  exception Empty
  val empty = []
  fun isEmpty (s:'a stack):bool =
    (case s
       of [] => true
        | _ => false)
  fun top (s:'a stack): =
    (case s
       of [] => raise Empty
        | x::xs => x)
  fun pop (s:'a stack):'a stack =
    (case s
        of [] => raise Empty
         | x::xs => xs)
  fun push (s:'a stack,x: 'a):'a stack = x::s
  fun rev (s:'a stack):'a stack = rev (s)
end
```

99 Bottles of Beer in Haskell

```
bottles :: Int -> String
bottles n
  | n == 0 = "no more bottles"
  | n == 1 = "1 bottle"
  | n >  1 = show n ++ " bottles"

verse :: Int -> String
verse n
  | n == 0 = "No more bottles of beer on the wall, "
             ++ "no more bottles of beer.\n"
             ++ "Go to the store and buy some more, "
             ++ "99 bottles of beer on the wall."
  | n > 0  = bottles n ++ " of beer on the wall, "
             ++ bottles n
             ++ " of beer.\n"
             ++ "Take one down and pass it around, "
             ++ bottles (n-1) ++ " of beer on the wall.\n"

main       = mapM (putStrLn . verse) [99,98..0]
```

Simon Johansson,

http://www.99-bottles-of-beer.net/language-haskell-1613.html

99 Bottles of Beer in Ha

```
bottles :: Int -> String
bottles n
  | n == 0 = "no more bottle
  | n == 1 = "1 bottle"
  | n >  1 = show n ++ " bot

verse :: Int -> String
verse n
  | n == 0 = "No more bottle
             ++ "no more bot
             ++ "Go to the s
             ++ "99 bottles
  | n > 0  = bottles n ++ "
             ++ bottles n
             ++ " of beer.\r
             ++ "Take one do
             ++ bottles (n-1

main       = mapM (putStrLn .
```

Peyton Jones et al., 1990

Functional

Pure: no side-effects

Lazy: computation only on demand; infinite data structures

Statically typed; types inferred

Algebraic data types, pattern matching, lists, strings

Great for compilers, domain-specific languages, type system research

Related to ML, OCaml

Simon Johansson,

http://www.99-bottles-of-beer.net/language-haskell-1613.html

sh, awk, perl, tcl, python, php

Scripting languages: glue for binding the universe together

```
class() {
  classname=`echo "$1" | sed -n '1 s/ *:.*$//p'`
  parent=`echo "$1" | sed -n '1 s/^.*: *//p'`
  hppbody=`echo "$1" | sed -n '2,$p'`

  forwarddefs="$forwarddefs
  class $classname;"

  if (echo $hppbody | grep -q "$classname()"); then
    defaultconstructor=
  else
    defaultconstructor="$classname() {}"
  fi
}
```

99 Bottles of Beer in AWK

```
BEGIN {
   for(i = 99; i >= 0; i--) {
      print ubottle(i), "on the wall,", lbottle(i) "."
      print action(i), lbottle(inext(i)), "on the wall."
      print
   }
}
function ubottle(n) {
   return sprintf("%s bottle%s of beer", n?n:"No more", n-1?"s":"")
}
function lbottle(n) {
   return sprintf("%s bottle%s of beer", n?n:"no more", n-1?"s":"")
}
function action(n) {
   return sprintf("%s", n ? "Take one down and pass it around," : \
                          "Go to the store and buy some more,")
}
function inext(n) {
   return n ? n - 1 : 99
}
```

OsamuAoki,
http://www.99-bottles-of-beer.net/language-awk-1623.html

99 Bottles of Beer in AWK

```
BEGIN {
   for(i = 99; i >= 0; i--) {
      print ubottle(i), "on the wall,", lbottle(i) "."
      print action(i), lbottle(inext(i)), "on the wall."
      print
   }
}
function ubottle(n) {
   return sprintf("%s bottle
}
function lbottle(n) {
   return sprintf("%s bottle
}
function action(n) {
   return sprintf("%s", n ?

}
function inext(n) {
   return n ? n - 1 : 99
}
```

Aho, Weinberger, and Kernighan, Bell Labs, 1977

Interpreted domain-specific scripting language for text processing

Pattern-action statements matched against input lines

C-inspired syntax

Automatic garbage collection

OsamuAoki,
http://www.99-bottles-of-beer.net/language-awk-1623.html

AWK (bottled version)

```
BEGIN{
    split( \
    "no mo"\
    "rexxN"\
    "o mor"\
    "exsxx"\
    "Take "\
   "one dow"\
  "n and pas"\
 "s it around"\
 ", xGo to the "\
"store and buy s"\
"ome more, x bot"\
"tlex of beerx o"\
"n the wall" , s,\
"x"); for( i=99 ;\
i>=0; i--){ s[0]=\
s[2] = i ; print \
s[2 + !(i) ] s[8]\
s[4+ !(i-1)] s[9]\
s[10]", " s[!(i)]\
s[8] s[4+ !(i-1)]\
s[9]".";i?s[0]--:\
s[0] = 99; print \
s[6+!i]s[!(s[0])]\
s[8] s[4 +!(i-2)]\
s[9]s[10] ".\n";}}
```

Wilhelm Weske,
http://www.99-bottles-of-beer.net/language-awk-1910.html

99 Bottles of Beer in Python

```
for quant in range(99, 0, -1):
   if quant > 1:
      print quant, "bottles of beer on the wall,", \
            quant, "bottles of beer."
      if quant > 2:
         suffix = str(quant - 1) + " bottles of beer on the wall."
      else:
         suffix = "1 bottle of beer on the wall."
   elif quant == 1:
      print "1 bottle of beer on the wall, 1 bottle of beer."
      suffix = "no more beer on the wall!"
   print "Take one down, pass it around,", suffix
   print ""
```

Gerold Penz,
http://www.99-bottles-of-beer.net/language-python-808.html

99 Bottles of Beer in Python

```
for quant in range(99, 0, -1
   if quant > 1:
      print quant, "bottles
            quant, "bottles
      if quant > 2:
         suffix = str(quant
      else:
         suffix = "1 bottle
   elif quant == 1:
      print "1 bottle of bee
      suffix = "no more beer
   print "Take one down, pas
   print ""
```

Guido van Rossum, 1989

Object-oriented, imperative

General-purpose scripting language

Indentation indicates grouping

Dynamically typed

Automatic garbage collection

Gerold Penz,
http://www.99-bottles-of-beer.net/language-python-808.html

99 Bottles of Beer in FORTH

```
: .bottles ( n -- n-1 )
   dup 1 = IF  ." One bottle of beer on the wall," CR
               ." One bottle of beer," CR
               ." Take it down,"
   ELSE  dup . ." bottles of beer on the wall," CR
         dup . ." bottles of beer," CR
         ." Take one down,"
   THEN
   CR
   ." Pass it around," CR
   1-
   ?dup IF  dup 1 = IF  ." One bottle of beer on the wall;"
            ELSE  dup . ." bottles of beer on the wall;"
            THEN
        ELSE  ." No more bottles of beer on the wall."
   THEN
   CR
;
: nbottles ( n -- )
  BEGIN  .bottles  ?dup NOT UNTIL ;

99 nbottles
```

Dan Reish,
http://www.99-bottles-of-beer.net/language-forth-263.html

99 Bottles of Beer in FORTH

```
: .bottles ( n -- n-1 )
   dup 1 = IF  ." One bottle
               ." One bottle
               ." Take it do
   ELSE  dup . ." bottles of
         dup . ." bottles of
         ." Take one down,"
   THEN
   CR
   ." Pass it around," CR
   1-
   ?dup IF  dup 1 = IF  ." O
            ELSE  dup . ." b
            THEN
        ELSE  ." No more bot
   THEN
   CR
;
: nbottles ( n -- )
  BEGIN  .bottles  ?dup NOT

99 nbottles
```

Moore, NRAO, 1973

Stack-based imperative language

Trivial, RPN-inspired grammar

Easily becomes cryptic

Untyped

Low-level, very lightweight

Highly extensible: easy to make programs compile themselves

Used in some firmware boot systems (Apple, IBM, Sun)

Inspired the PostScript language for laser printers

Dan Reish,
http://www.99-bottles-of-beer.net/language-forth-263.html

The Whitespace Language

Edwin Brady and Chris Morris, April 1st, 2003

Imperative, stack-based language

Space, Tab, and Line Feed characters only

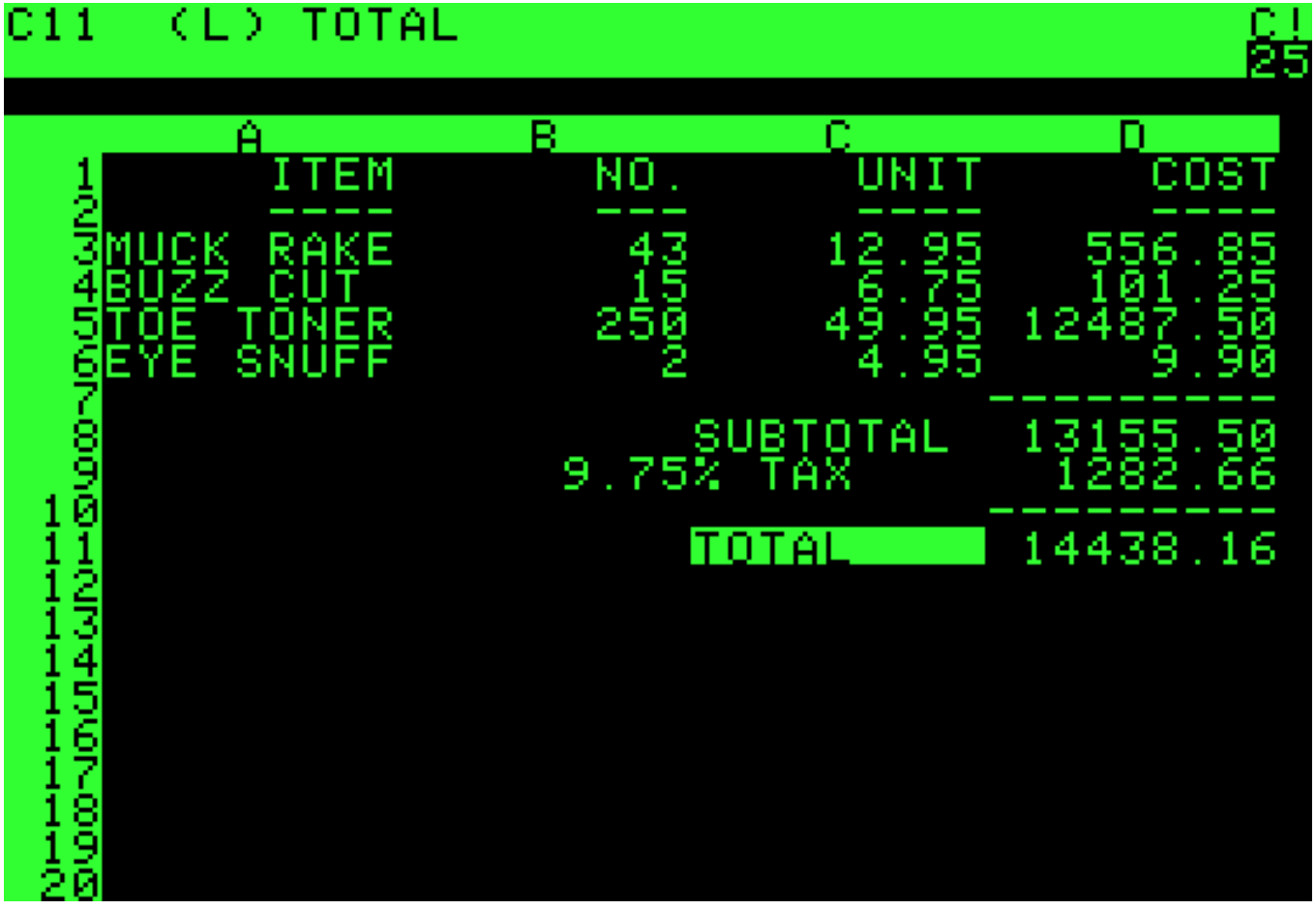
Number literals in binary: Space=0, Tab=1, LF=end

Less-than-programmer-friendly syntax; reduces toner consumption

Andrew Kemp, `http://compsoc.dur.ac.uk/whitespace/`

VisiCalc, Lotus 1-2-3, Excel

The spreadsheet style of programming



Visicalc on the Apple II, c. 1979

SQL

Database queries

```
CREATE TABLE shirt (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    style ENUM('t-shirt', 'polo', 'dress') NOT NULL,
    color ENUM('red', 'blue', 'white', 'black') NOT NULL,
    owner SMALLINT UNSIGNED NOT NULL
          REFERENCES person(id),
    PRIMARY KEY (id)
);

INSERT INTO shirt VALUES
(NULL, 'polo', 'blue', LAST_INSERT_ID()),
(NULL, 'dress', 'white', LAST_INSERT_ID()),
(NULL, 't-shirt', 'blue', LAST_INSERT_ID());
```

From thinkgeek.com

99 Bottles of Beer in SQL

```
SELECT
  CASE (bottlecount)
    WHEN 0 THEN 'No more bottle of beer on the wall, no more bottles o
                'Go to the store and buy some more, 99 bottles of beer
    WHEN 1 THEN '1 bottle of beer on the wall, 1 bottle of beer. ' ||
                'Take one down and pass it around, no more bottles of
    WHEN 2 THEN '2 bottles of beer on the wall, 2 bottles of beer. ' |
                'Take one down and pass it around, 1 bottle of beer on
    ELSE
      rtrim (cast((BottleCount) as char(2))) || ' bottles of beer on t
      rtrim (cast((BottleCount) as char(2))) || ' bottles of beer. ' |
      'Take one down and pass it around, ' ||
      rtrim (cast((BottleCount)-1 as char(2))) || ' bottles of beer on
  END
FROM
(
  SELECT avalue * 10 + bvalue as bottlecount
  FROM
    (VALUES (9), (8), (7), (6), (5), (4), (3), (2), (1), (0)) a(avalue
    (VALUES (9), (8), (7), (6), (5), (4), (3), (2), (1), (0)) b(bvalue
) as valuelist;
```

Kent Olsen,
http://www.99-bottles-of-beer.net/language-sql-967.html

99 Bottles of Beer in SQL

```
SELECT
  CASE (bottlecount)
    WHEN 0 THEN 'No more bottle of beer on the wall, no more bottles o
                'Go to the store and buy some more, 99 bottles of beer
    WHEN 1 THEN '1 bottle of beer on the wall, 1 bottle of beer. ' ||
                'Take one down and pass it around, no more bottles of
    WHEN 2 THEN '2 bottles of beer on the wall, 2 bottles of beer. ' |
                'Take one do                                          n
    ELSE
      rtrim (cast((BottleCou                                          t
      rtrim (cast((BottleCou                                          |
      'Take one down and pas
      rtrim (cast((BottleCou                                          n
  END
FROM
(
  SELECT avalue * 10 + bvalu
  FROM
    (VALUES (9), (8), (7), (                                          e
    (VALUES (9), (8), (7), (                                          e
) as valuelist;
```

Chamberlin and Boyce, IBM, 1974

Declarative language for databases

Semantics based on the relational model

Queries on tables: select with predicates, joining, aggregating

Database query optimization: declaration to procedure

Kent Olsen,
http://www.99-bottles-of-beer.net/language-sql-967.html

Prolog

Logic Language

```
witch(X)  <= burns(X) and female(X).
burns(X)  <= wooden(X).
wooden(X) <= floats(X).
floats(X) <= sameweight(duck, X).

female(girl).           {by observation}
sameweight(duck,girl). {by experiment }

? witch(girl).
```

99 Bottles of Beer in Prolog

```
bottles :-
    bottles(99).

bottles(1) :-
    write('1 bottle of beer on the wall, 1 bottle of beer,'), nl,
    write('Take one down, and pass it around,'), nl,
    write('Now they are all gone.'), nl,!.
bottles(X) :-
    write(X), write(' bottles of beer on the wall,'), nl,
    write(X), write(' bottles of beer,'), nl,
    write('Take one down and pass it around,'), nl,
    NX is X - 1,
    write(NX), write(' bottles of beer on the wall.'), nl, nl,
    bottles(NX).
```

Remko Trocon et al.,
http://www.99-bottles-of-beer.net/language-prolog-965.html

99 Bottles of Beer in Prolog

```
bottles :-
    bottles(99).

bottles(1) :-
    write('1 bottle of beer
    write('Take one down, ar
    write('Now they are all
bottles(X) :-
    write(X), write(' bottle
    write(X), write(' bottle
    write('Take one down and
    NX is X - 1,
    write(NX), write(' bottl
    bottles(NX).
```

Alain Colmerauer et al., 1972

Logic programming language

Programs are relations: facts and rules

Program execution consists of trying to satisfy queries

Designed for natural language processing, expert systems, and theorem proving

Remko Trocon et al.,
http://www.99-bottles-of-beer.net/language-prolog-965.html