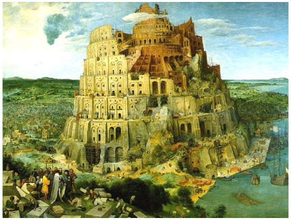
Programming Languages and Translators

Stephen A. Edwards

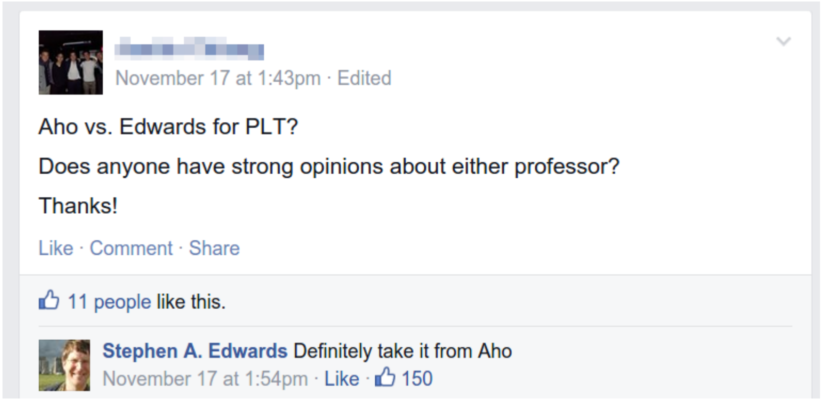
Columbia University

Spring 2021



Pieter Bruegel, *The Tower of Babel*, 1563

Facebook on 4115



[redacted]

November 17 at 1:43pm · Edited

Aho vs. Edwards for PLT?

Does anyone have strong opinions about either professor?

Thanks!

11 people like this.

Stephen A. Edwards Definitely take it from Aho

November 17 at 1:54pm · Like · 150

Sadly, Aho has retired from teaching 4115.

But now, Prof. Baishakhi Rey and Prof. Ronghui Gu also teach 4115.

Instructor

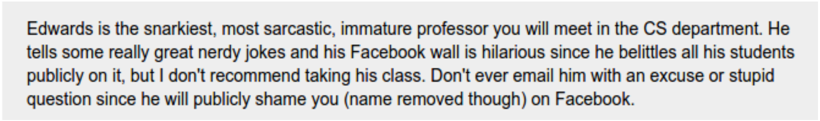
Prof. Stephen A. Edwards

sedwards@cs.columbia.edu

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

My Zoom office hours will be posted on Courseworks

Culpa on Edwards


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

Recommended Text

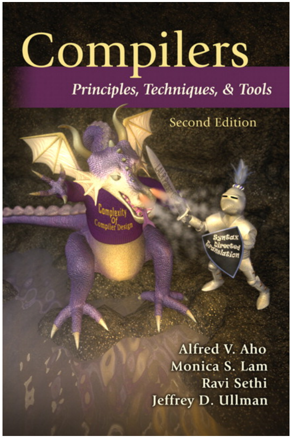
Alfred V. Aho, Monica S. Lam,
Ravi Sethi, and Jeffrey D. Ullman.

Compilers: Principles, Techniques, and Tools.

Addison-Wesley, 2006.
Second Edition.

Bug Al about all bugs.

You can get away with the first edition.


Compilers
Principles, Techniques, & Tools
Second Edition
Complexity of Compiler Design
Syntax Directed Translation
Alfred V. Aho
Monica S. Lam
Ravi Sethi
Jeffrey D. Ullman

Assignments and Grading

40%	Team Programming Project
20%	Midterm Exam
30%	Final Exam (cumulative)
10%	Three individual homework assignments
0%	Effort*

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

*Do or do not; there is no try —Yoda

Schedule

Lectures: Mondays and Wednesdays, 5:40 – 6:55 PM

Via Zoom; link on Courseworks

January 11 – April 14th

Midterm Exam	February 25
Final Exam	April 16
Presentations	April 23*
Final Team project reports	April 23

* You can present before April 23. All team members must present.

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 coding

COMS W3261 Computer Science Theory

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

Collaboration

Read the CS Department's Academic Honesty Policy:
`https://www.cs.columbia.edu/education/honesty/`

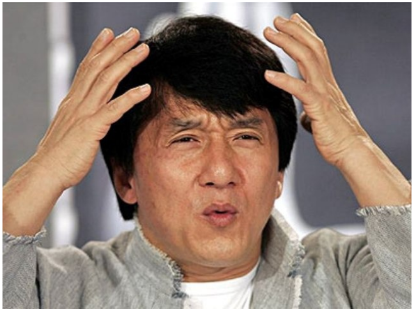
Collaborate with your team on the project.

Do your homework by yourself.

- OK: Discussing lecture content, OCaml features
- Not OK: Solving a homework problem with classmates
- Not OK: Posting any homework questions or solutions

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.

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



The Team Project

The Team 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, written in OCaml; generating LLVM
5. A final project report
6. A final project presentation

Teams

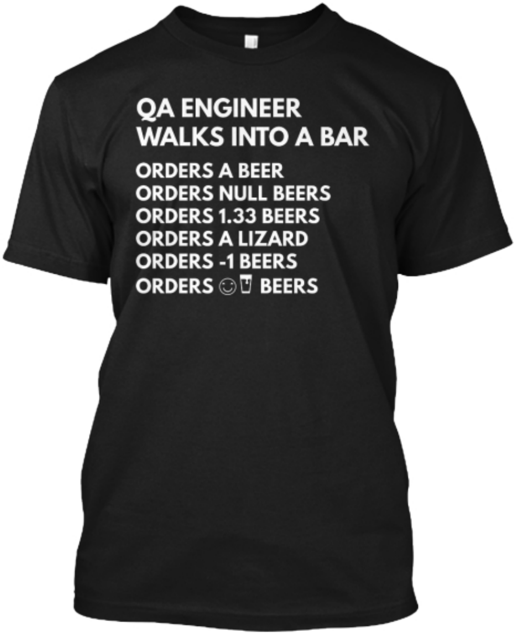
Immediately start forming four-person teams

Each team will develop its own language

Each teach member should participate in design, coding, testing, and documentation

Choose one team member to head specific tasks:

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



teespring.com

- Cover for flaky teammates. They will thank you later by completely reforming their behavior, making up for all the times you did their work for them.
- Assign the least qualified team member to each task.
- Avoid leadership; include every feature and make all decisions by arguing.
- Don’t let other members speak; they don’t want to.
- Ignore other members’ opinions: you’re always right; they’re always wrong.

- Never let anybody take responsibility for anything. Write software communally so nobody is ever at fault.
- Never tell the instructor or a TA that something is wrong with your group. It will only lower your grade.
- Implement your scanner completely before testing it or starting on the parser.
- Just do unit tests; when you put things together, everything will work fine.
- “This is like a Greek tragedy: you’re told everything that will happen, you think it won’t happen to me, then it happens anyway”

RED
FLAGS

Student Testimonials

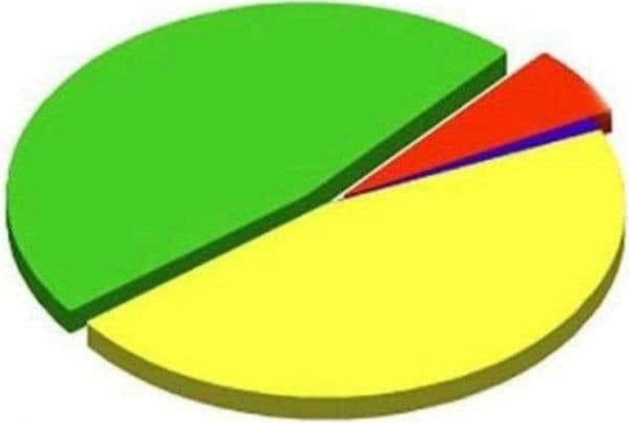
"START EARLY, and really be selective in picking your team. A bad team will ruin the semester for you."

"Start early and be sure to pester the TAs for help. Also, half of your team will be slackers and you will lose all faith in humanity."

"We didn't bring this up earlier since we imagined that when it became crunch time everyone in the group would take the project seriously, but that hasn't been the case."

EVERY GROUP PROJECT
DOES 99% OF THE WORK
HAS NO IDEA WHAT'S GOING ON THE WHOLE TIME
SAYS HE'S GOING TO HELP BUT HE'S NOT
DISAPPEAR AT THE VERY BEGINNING AND DOESN'T SHOW UP AGAIN TIL THE VERY END
IN SCHOOL YOU HAVE EVER DONE
WeKnowMemes

WHAT I LEARN FROM GROUP PROJECTS



🟥 The information
🟦 How to work with people
🟨 How to do entire projects on my own
🟩 How much I hate people

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

How Do You Work In a Team?

If I knew, I'd use the knowledge to take over the world

- Address problems sooner rather than later

  If you think your teammate's a flake, you're right

- Complain to me or your TA as early as possible

  Alerting me a day before the project is due isn't helpful

- Not every member of a team will get the same grade

  Remind your slacking teammates of this early and often

- I have forcibly split and dissolved teams

  If someone is really underperforming, dump his ass

What Google Learned From Its Quest to Build the Perfect Team

Things that *did not* matter

- ✖ Members’ intelligence
- ✖ Members’ experience
- ✖ Mix of personality types
- ✖ Whether the members were close friends
- ✖ Strong organization
- ✖ Gender balance

`http://mobile.nytimes.com/2016/02/28/magazine/what-google-learned-from-its-quest-to-build-the-perfect-team.html`

`https://hunterwalk.com/2016/09/03/google-finds-that-successful-teams-are-about-norms-not-just-smarts/`

What Google Learned From Its Quest to Build the Perfect Team

Things that *did* matter

Team “norms.” Unwritten rules of team interaction.

- ✓ That every team member spoke in the same proportion
- ✓ That team members had “social sensitivity”
  Empathy for fellow team members: the ability to read others’ feelings through void, expressions, etc.

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

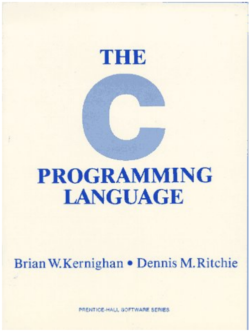
Include the source code for an interesting program in your language

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 Language*; see the class website).


THE
C
PROGRAMMING
LANGUAGE
Brian W. Kernighan • Dennis M. Ritchie
PRENTICE-HALL SOFTWARE SERIES

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	February 3 soon
Language Reference Manual and parser	February 22
Hello World Demo	March 24
Final Report	April 23

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

Music manipulation language

Mathematical function manipulator

Simple scripting language (à lá Tcl)

Two Common Mistakes to Avoid

Configuration File Syndrome

- Your language should have more than just nouns
- Must be able to express *algorithms*, not just data

Standard Library Syndrome

- Good languages enable you to *build* abstractions, not just *provide* them
- Write your standard library in your language
- Aim for Legos, not Microsoft Word

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 generate LLVM code

- Compilers should lower the level of abstraction; LLVM provides a machine-independent, low-level IR.
- Robust, widespread “collection of modular and reusable compiler and toolchain technologies.”

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:

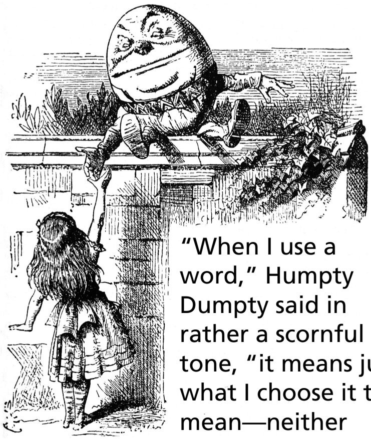
$$
\begin{array}{rcl}
\mathit{expr} & \rightarrow & \mathit{expr} + \mathit{expr} \\
 & | & \mathit{expr} - \mathit{expr} \\
 & | & \mathit{expr} * \mathit{expr} \\
 & | & \mathit{expr} \,/\, \mathit{expr} \\
 & | & (\, \mathit{expr}\,) \\
 & | & \textbf{digits}
\end{array}
$$

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

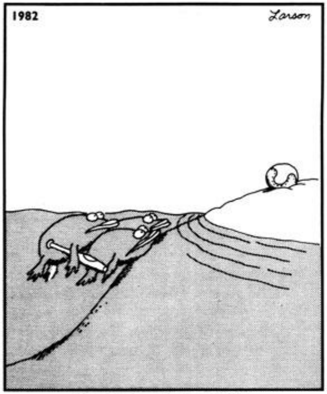
Nonsensical in Java:

```
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
```

FORTRAN

Before

Backus, IBM, 1956
Imperative language for science and engineering
First compiled language
Fixed format punch cards
Arithmetic expressions, If, Do, and Goto statements
Scalar and array types
Limited string support
Still common in high-performance computing
Inspired most modern languages, especially BASIC

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
```

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).
```

English-like syntax: 300 reserved words
Grace Hopper et al.

LISP, Scheme, Common LISP

Functional, high-level languages

```
(defun append (l1 l2)
  (if (null l1)
      l2
      (cons (first l1) (append (rest l1) l2))))
```

LISP, Scheme, Common LISP

Functional, high-level languages

```
(defun append (l1 l2)
  (if (null l1)
      l2
      (cons (first l1) (app
```

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

APL

Powerful operators, interactive, custom character set

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

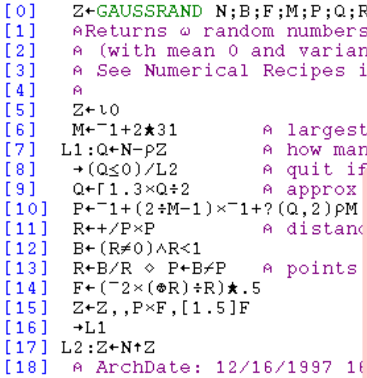
“Emoticons for Mathematicians”

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

At right: Datamedia APL Keyboard

APL

Powerful operators, interactive, custom character set



```
 Z←GAUSSRAND N;B;F;M;P;Q;R
 ⍝Returns ⍵ random numbers having a Gaussian normal distribution
 ⍝ (with mean 0 and variance 1)  Uses the Box-Muller method.
 ⍝ See Numerical Recipes in C, pg. 289.
 ⍝
 Z←⍳0
 M←¯1+2⋆31         ⍝ largest integer
L1:Q←N-⍴Z          ⍝ how many more we need
 →(Q≤0)/L2         ⍝ quit if
 Q←⌈1.3×Q÷2        ⍝ approx
 P←¯1+(2÷M-1)×¯1+?(Q,2)⍴M
 R←+/P×P           ⍝ distanc
 B←(R≠0)∧R<1
 R←B/R ⋄ P←B⌿P     ⍝ points
 F←(¯2×(⍟R)÷R)⋆.5
 Z←Z,,P×F,[1.5]F
 →L1
L2:Z←N↑Z
 ⍝ ArchDate: 12/16/1997 16
```

"Emoticons for Mathematicia

Source: Jim Weigang, http://www.chilton.com/~j

At right: Datamedia APL Keyboard

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

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
```

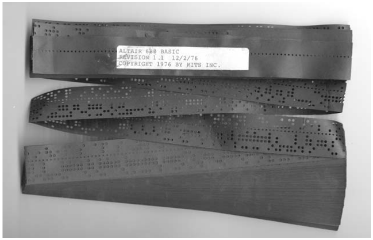
SNOBOL: Parse IBM 360 assembly. From Gimpel's book, http://www.snobol4.org/

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

```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(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(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`

Efficiency for systems programming

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

C

Efficiency for systems prog

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

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

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

```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 Haskell

```
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.\n
             ++ "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 of beer on the wall,", \
            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 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
         ." 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`

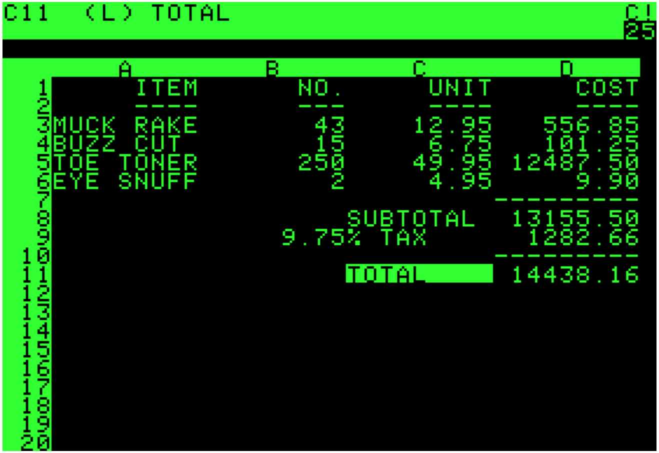
The Whitespace Language

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



```
C11  (L) TOTAL                                          C!
                                                        25

             A           B           C           D
 1          ITEM        NO.        UNIT        COST
 2          ----        ---        ----        ----
 3MUCK RAKE              43       12.95      556.85
 4BUZZ CUT               15        6.75      101.25
 5TOE TONER             250       49.95    12487.50
 6EYE SNUFF               2        4.95        9.90
 7                                       ----------
 8                          SUBTOTAL       13155.50
 9                      9.75% TAX           1282.66
10                                       ----------
11                          TOTAL          14438.16
12
13
14
15
16
17
18
19
20
```

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());
```

SQL

Database queries

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

INSERT INTO shirt VALUES
(NULL, 'polo', 'blue', LAST
(NULL, 'dress', 'white', LA
(NULL, 't-shirt', 'blue', L
```

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

```
> SELECT * FROM users WHERE clue > 0
0 rows returned
```

From thinkgeek.com

Prolog

Logic Language

```
witch(X)  <= burns(X), 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).
```

Prolog

Logic Language

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

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

? witch(girl).
```

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