Programming Languages and Translators

Ronghui Gu

Spring 2019

Columbia University

Prof. Ronghui Gu

515 Computer Science Building

ronghui.gu@columbia.edu

Office hours: Thursdays 1:30 - 2:30 PM / by appointment

Prof. Stephen A. Edwards and Prof. Baishakhi Rey also teach 4115.
*These slides are borrowed from Prof. Edwards.

What is a Programming Language?

A programming language is a notation that a person and a computer can both understand.

- It allows you to express what is the **task** to do
- It allows a computer to **execute** the task

Every programming language has a **syntax** and **semantics**.

- Syntax: how characters combine to form a program
- Semantics: what's the task that the program *means*

How characters combine to form a program.

Calculate the n-th Fibonacci number.

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.

Usually done with a context-free grammar.

Typical syntax for algebraic expressions:

$$
\begin{aligned}
expr \quad \rightarrow \quad & expr + expr \\
| \quad & expr - expr \\
| \quad & expr * expr \\
| \quad & expr \: / \: expr \\
| \quad & (\: expr \:) \\
| \quad & \textbf{digits}
\end{aligned}
$$

What a well-formed program "means."

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

```c
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;
}
```

Something may be syntactically correct but semantically nonsensical

> *The rock jumped through the hairy planet.*

Or ambiguous

> *The chickens are ready to eat.*

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; }
}
```

A programming language is a notation that a person and a computer can both understand.

- It allows you to express what is the **task** to do
- It allows a computer to **execute** the task

A translator translates what you express to what a computer can execute.

What is a Translator?

C

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

Assembly

```
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
```

Bytes

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

Course Structure

Course Structure

Course home page:
https://www.cs.columbia.edu/ rgu/courses/4115/spring2019

26 Lectures: Mondays and Wednesdays, 2:40 – 3:55 PM

Jan 23 – May 6

833 Seeley W. Mudd Building

Team project (presentation & report)	May 15*
Midterm Exam	Mar 13
Final Exam	May 6
3 Assignments	

* You can present before May 15. All team members must present.

Assignments and Grading

40% Team Programming Project

20% Midterm Exam

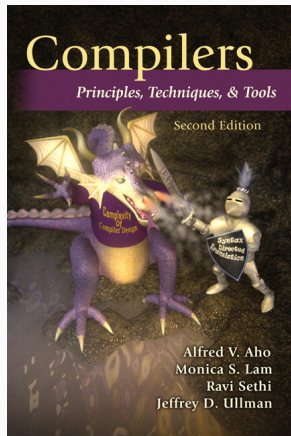20% Final Exam (cumulative)

20% Three individual homework assignments

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

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

Compilers: Principles, Techniques, and Tools.

Addison-Wesley, 2006.
Second Edition.

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)

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)

The Team Project

The Team Project

Design and implement your own little language.

Six deliverables:

1. A proposal describing your language (due Feb 11)
2. A language reference manual (due Feb 27)
3. A milestone: compiling "Hello World" (due Apr 1)
4. A compiler for it, written in OCaml; generating LLVM
5. A final project report (due May 15)
6. A final project presentation (due May 15)

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

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

START EARLY!

How Do You Work In a Team?

- 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*

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

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

- 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).
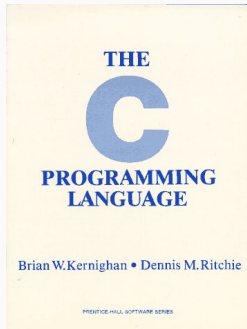
Final Report Sections

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	Feb 11 soon
Language Reference Manual and Parser	Feb 27
Hello World Demo	Apr 1
Final Report and Presentation	May 15

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)

Three 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

C-to-C Translator Syndrome

- Your compiler's output should not look like its input

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."

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
```

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
```

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

B

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
```

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) .
```
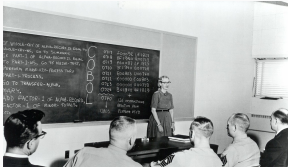

English-like syntax: 300 reserved words
Grace Hopper et al.

Functional, high-level languages

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

Functional, high-level langu

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

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

Powerful operators, interactive, custom character set

```
[0]     Z←GAUSSRAND N;B;F;M;P;Q;R
[1]     ⍝Returns ω 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-ρZ          ⍝ 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)ρM  ⍝ 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

32

Powerful operators, interactive, custom character set

```
[0]    Z←GAUSSRAND N;B;F;M;P;Q;
[1]    ⍝Returns ω random numbers
[2]    ⍝ (with mean 0 and varian
[3]    ⍝ See Numerical Recipes :
[4]    ⍝
[5]    Z←⍳0
[6]    M←¯1+2⋆31        ⍝ larges⊺
[7]  L1:Q←N-⍴Z          ⍝ how mar
[8]    →(Q≤0)/L2        ⍝ quit i⍦
[9]    Q←⌈1.3×Q÷2       ⍝ approx
[10]   P←¯1+(2÷M-1)×¯1+?(Q,2)⍴M
[11]   R←+/P×P          ⍝ distanc
[12]   B←(R≠0)∧R<1
[13]   R←B/R ◇ P←B≠P    ⍝ points
[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 1⣿
```

"Emoticons for Mathematician

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

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

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);
```

33

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
```

34

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.
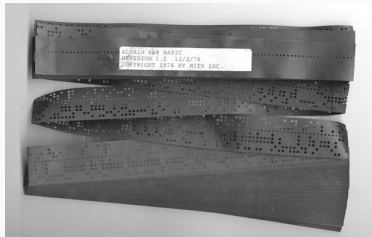
The object-oriented philosophy

```
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;
```

```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 wa
      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.\
        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
    }
  }
}
```

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

99 Bottles of Beer in Java

```java
class Bottles {
  public static void main
    String s = "s";
    for (int beers=99; b
      System.out.print(b
      System.out.println
      if (beers==0) {
        System.out.print
        System.out.print
        System.exit(0);
      } else
        System.out.print
      s = (--beers == 1)
      System.out.println
    }
  }
}
```

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

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

C

Efficiency for systems progr

```c
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

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
```

39

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,

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

verse :: Int -> String
verse n
  | n == 0 = "No more be
             ++ "no more
             ++ "Go to t
             ++ "99 bott
  | n > 0  = bottles n +
             ++ bottles
             ++ " of bee
             ++ "Take on
             ++ bottles

main       = mapM (putStr
```

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,

Scripting languages: glue for binding the universe together

```sh
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

```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?
}
function lbottle(n) {
    return sprintf("%s bottle%s of beer", n?n:"no more", n-1?
}
function action(n) {
    return sprintf("%s", n ? "Take one down and pass it aroun
                             "Go to the store and buy some mo
}
function inext(n) {
    return n ? n - 1 : 99
}
```

99 Bottles of Beer in AWK

```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 bo                          ?
}
function lbottle(n) {
    return sprintf("%s bo                          ?
}
function action(n) {
    return sprintf("%s",
                                                    10
                                                    10
}
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

42

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

```
     sprit ( " \
     " 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)] \
```

```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
        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

```python
for quant in range(99, 0, -1):
    if quant > 1:
        print quant, "bott
              quant, "bott
        if quant > 2:
            suffix = str(qu
        else:
            suffix = "1 bo
    elif quant == 1:
        print "1 bottle of
        suffix = "no more
    print "Take one down,
    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

```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
```

```forth
: .bottles ( n -- n-1 )
   dup 1 = IF    ." One b
                 ." One b
                 ." Take i
   ELSE  dup . ." bottle
         dup . ." bottle
         ." Take one dow
   THEN
   CR
   ." Pass it around," C
   1-
   ?dup IF   dup 1 = IF
             ELSE  dup .
             THEN
         ELSE   ." No more
   THEN
   CR
;
: nbottles ( n -- )
   BEGIN   .bottles  ?dup
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[45]

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/

The spreadsheet style of programming

Visicalc on the Apple II, c. 1979

Database queries

```sql
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());
```

Database queries

```
CREATE TABLE shirt (
    id SMALLINT UNSIGNED
    style ENUM('t-shirt
    color ENUM('red', '
    owner SMALLINT UNSIGNED
            REFERENCES pe
    PRIMARY KEY (id)
);

INSERT INTO shirt VALUE
(NULL, 'polo', 'blue', )
(NULL, 'dress', 'white'
(NULL, 't-shirt', 'blue
```

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

From thinkgeek.com

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).
```

Logic Language

```
witch(X)   <= burns(X), female(X).
burns(X)   <= wooden(X).
wooden(X)  <= floats(X).
floats(X)  <= sameweight
                          Alain Colmerauer et al., 1972
female(girl).
sameweight(duck, girl).  Logic programming language

? witch(girl).           Programs are relations: facts and
                         rules
```

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