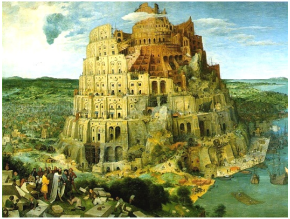
Programming Languages and Translators

Stephen A. Edwards

Columbia University

Fall 2011



Pieter Bruegel, *The Tower of Babel*, 1563

Instructor

Prof. Stephen A. Edwards

sedwards@cs.columbia.edu

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

462 Computer Science Building

Schedule

Mondays and Wednesdays, 4:10 – 5:25 PM

501 Northwest Corner Building

Lectures: September 7 to December 7

Midterm: November 2

Final: December 12 (in-class)

Final project report: December 22

Holidays: November 7 (Election day)

Objectives

Theory of language design

- Finer points of languages
- Different languages and paradigms

Practice of Compiler Construction

- Overall structure of a compiler
- Automated tools and their use
- Lexical analysis to assembly generation

Required Text

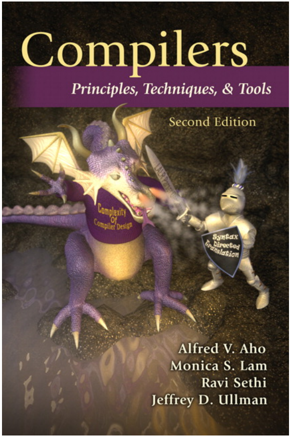
Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman.

Compilers: Principles, Techniques, and Tools.

Addison-Wesley, 2006. Second Edition.

Bug Al about all bugs.

You can get away with the first edition.


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% Programming Project

20% Midterm

30% Final

10% Individual homework

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

Prerequisites

COMS W3157 Advanced Programming

- Teams will build a large software system
- Makefiles, version control, test suites
- Testing will be as important as development

COMS W3261 Computability and Models of Computation

- You need to understand grammars
- We will be working with regular and context-free languages

Class Website

Off my home page, http://www.cs.columbia.edu/~sedwards/

Contains syllabus, lecture notes, and assignments.

Schedule will be continually updated during the semester.

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 cheat on assignments (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 people cheating and sent them to the dean. Please try to break my streak.

Part I

The Project

The Project

Design and implement your own little language.

Five deliverables:

1. A proposal describing and motivating your language
2. A language reference manual defining it formally
3. A compiler or interpreter for your language running on some sample programs
4. A final project report
5. A final project presentation

Teams

Immediately start forming four-person teams to work on this project.

Each team will develop its own langauge.

All members of the team should be familiar with the whole project.

First Three Tasks

1. Decide who you will work with
   You'll be stuck with them for the term; choose wisely.
2. Elect a team leader
   Languages come out better from dictatorships, not democracies. Besides, you'll have someone to blame.
3. Select a weekly meeting time
   Harder than you might think. Might want to discuss with a TA you'd like to have so it is convenient for him/her as well.

Project Proposal

Describe the language that you plan to implement.

Explain what problem your language can solve and how it should be used.

Describe an interesting, representative program in your language.

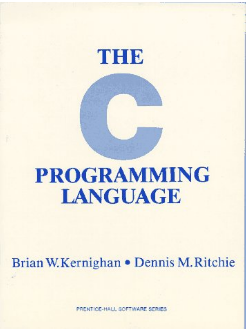
Give some examples of its syntax and an explanation of what it does.

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).


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

Final Report Sections

1. Introduction: the proposal
2. Language Tutorial
3. Language Reference Manual
4. Project Plan
5. Architectural Design
6. Test Plan
7. Lessons Learned
8. Complete listing

Due Dates

Proposal	September 28 soon
Reference Manual	October 31
Final Report	December 22

Design a language?

A small, domain-specific language.

Think of awk or php, not Java or C++.

Examples from earlier terms:

Geometric figure drawing language

Matlab-like array manipulation language

Quantum computing language

Screenplay animation language

Escher-like pattern generator

Music manipulation language (harmony)

Web surfing language

Mathematical function manipulator

Simple scripting language (à lá Tcl)

Two Common Mistakes to Avoid

Configuration File Syndrome

- Your language must be able to express *algorithms*, not just data
- If your language looks like “a bird and a bird and a turtle and a pond and grass and a rock,” it has fallen victim to configuration file syndrome and needs to be changed

Standard Library Syndrome

- The beauty of a language is its ability to express many different things by combining only a few
- The standard library supplied by your language should be small or nonexistent. Instead, think about how you could express your standard library in your language.
- Aim for Legos, not Microsoft Word

Part II

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:

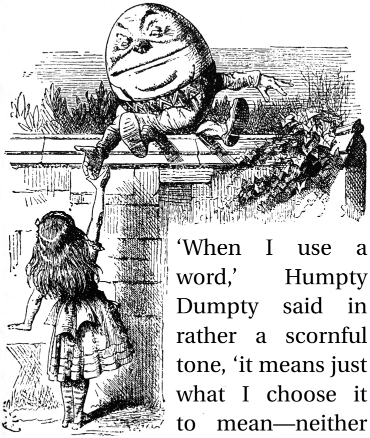
$$\begin{array}{rcl} \mathit{expr} & \rightarrow & \mathit{expr} + \mathit{expr} \\ & | & \mathit{expr} - \mathit{expr} \\ & | & \mathit{expr} * \mathit{expr} \\ & | & \mathit{expr} \,/\, \mathit{expr} \\ & | & \mathbf{digit} \\ & | & (\mathit{expr}) \end{array}$$

Components of a language: Semantics

What a well-formed program "means."

The semantics of C says this computes the *n*th 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; }
}
```

Specifying Semantics

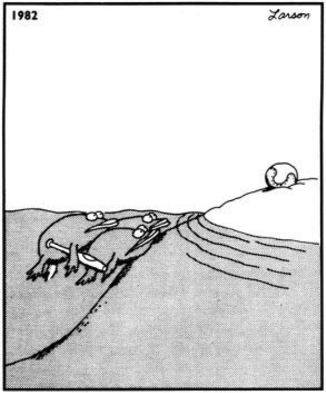
Doing it formally is beyond the scope of this class, but there are basically two ways:

- Operational semantics
  Define a virtual machine and how executing the program evolves the state of the virtual machine
- Denotational semantics
  Shows how to build the function representing the behavior of the program (i.e., a transformation of inputs to outputs) from statements in the language.

Most language definitions use an informal operational semantics written in English.

Part III

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
```

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).
```

From cafepress.com

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))
```

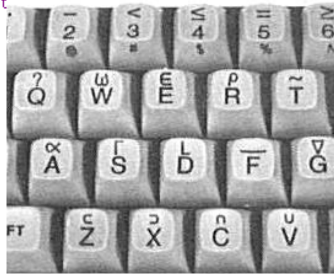
APL

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
[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
```

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

At right: Datamedia APL Keyboard



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
```

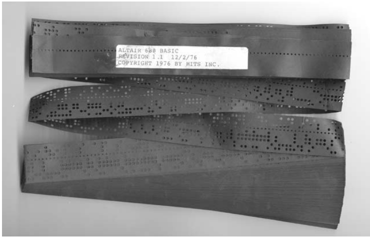
SNOBOL: Parse IBM 360 assembly. From Gimpel's book, http://www.snobol4.org/

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
```

Started the whole Bill Gates/ Microsoft thing. BASIC was invented by Dartmouth researchers John George Kemeny and Thomas Eugene Kurtz.



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;
```

C

Efficiency for systems programming

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

ML, Miranda, Haskell

Functional languages with a 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
```

sh, awk, perl, tcl, python, php

Scripting languages: glue for binding the universe together

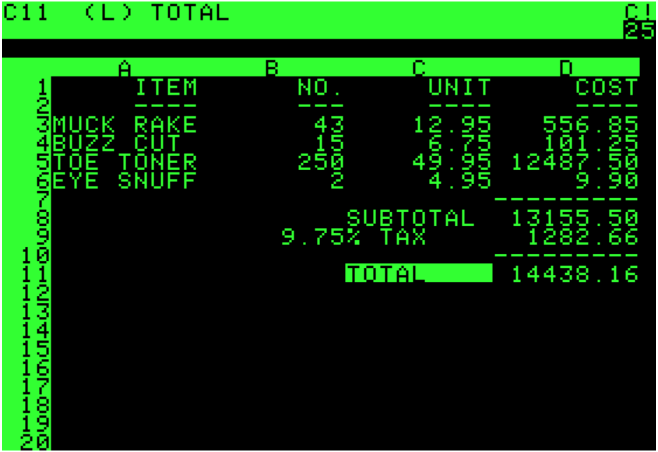
```
class() {
  classname=`echo "$1" | sed -n '1 s/ *:.*$//p'`
  parent=`echo "$1" | sed -n '1 s/^.*: *//p'`
  hppbody=`echo "$1" | sed -n '2,$p'`

  forwarddefs="$forwarddefs
  class $classname;"

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

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

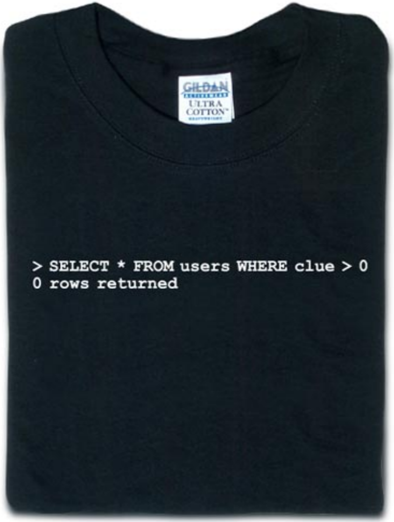
SQL

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 T-Shirt



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

From thinkgeek.com

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).
```