

Programming Languages and Translators

Stephen A. Edwards

Columbia University

Fall 2014



Pieter Bruegel, *The Tower of Babel*, 1563

Instructor and Schedule

Prof. Stephen A. Edwards

sedwards@cs.columbia.edu

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

462 Computer Science Building

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

September 3 – December 8

Midterm: Wednesday, October 29

Final: Monday, December 8

Presentations: December 15 – 17

Final project reports: December 17

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

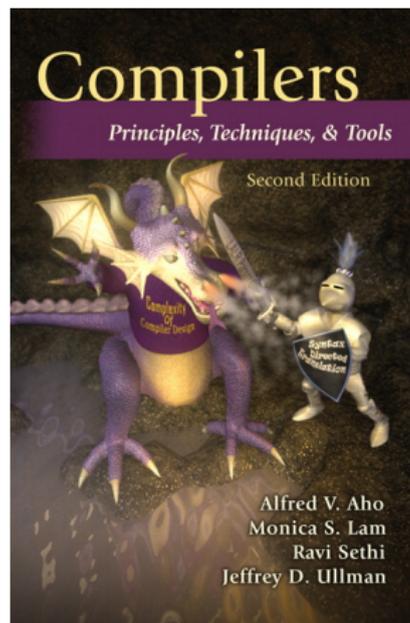
Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman.

Compilers: Principles, Techniques, and Tools.

Addison-Wesley, 2006. Second Edition.

Bug AI about all bugs.

You can get away with the first edition.



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

- ▶ 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 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 your language
2. A language reference manual defining it formally
3. A compiler for it, running sample programs
4. A final project report
5. A final project presentation

Teams

Immediately start forming four-person teams

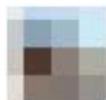
Each team will develop its own language

Assign each team member a specific role

Role	Responsibilities
Manager	Timely completion of deliverables
Language Guru	Language design
System Architect	Compiler architecture, environ.
Verification & Validation	Test plan, test suites

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.



about an hour ago

When I die, I would like the people I did group projects with to lower me in to my grave so they can let me down one last time.

lamebook.com

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

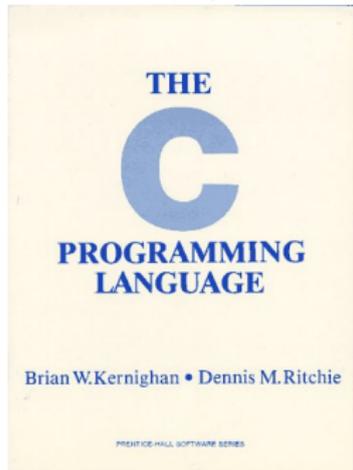
Include the source code for an interesting program in your language

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).



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

Due Dates

Proposal September 24 **soon**

Reference Manual October 27

Final Report December 17

Design a language?

A small, domain-specific language: 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)

Mathematical function manipulator

Simple scripting language (à la 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.

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:

$$\begin{aligned} \text{expr} &\rightarrow \text{expr} + \text{expr} \\ &| \text{expr} - \text{expr} \\ &| \text{expr} * \text{expr} \\ &| \text{expr} / \text{expr} \\ &| \mathbf{\text{digit}} \\ &| (\text{expr}) \end{aligned}$$

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.’

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:

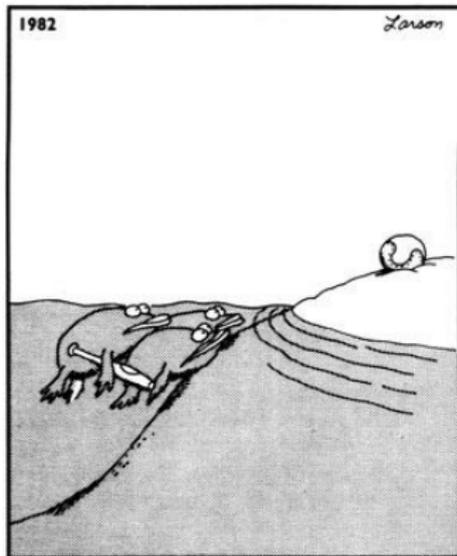
```
class Foo {  
    int bar(int x) { return Foo; }  
}
```

Ambiguous in Java:

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

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).
```



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-doinstert)
        (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 ⍝ A largest integer
[7] L1:Q←N-ρZ ⍝ A how many more we need
[8] →(Q≤0)/L2 ⍝ A quit if none
[9] Q←⌈1.3×Q÷2 ⍝ A approx num points needed
[10] P←⌈1+(2÷M-1)×⌈1+?(Q,2)ρM ⍝ A random points in -1 to 1 square
[11] R←+/P×P ⍝ A distance from origin squared
[12] B←(R≠0)∧R<1
[13] R←B/R ⍊ P←B÷P ⍝ A 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/gstrand.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;
  FI;
```

SNOBOL, Icon

String-processing languages

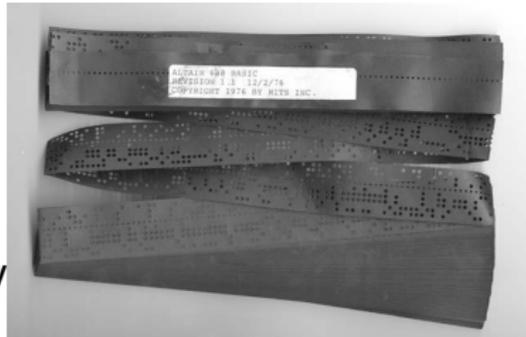
```
LETTER = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ$#@'  
SP.CH  = "+-,=.*()' /& "  
SCOTA  = SP.CH  
SCOTA  '&' =  
Q      = ""  
QLIT   = Q FENCE BREAK(Q) Q  
ELEM   = QLIT | 'L' Q | ANY(SCOTA) | BREAK(SCOTA) | REM  
F3     = ARBNO(ELEM FENCE)  
B      = (SPAN(' ') | RPOS(0)) FENCE  
F1     = BREAK(' ') | REM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

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;
```

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 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
```

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				C1
				25
	A	B	C	D
1	ITEM	NO.	UNIT	COST
2	---	---	---	---
3	MUCK RAKE	43	12.95	556.85
4	BUZZ CUT	15	6.75	101.25
5	TOE TONER	250	49.95	12487.50
6	EYE SNUFF	2	4.95	9.90
7				
8			SUBTOTAL	13155.50
9			9.75% TAX	1282.66
10			TOTAL	14438.16
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				

Visicalc on the Apple II, c. 1979

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

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).
```

