

## Programming Languages and Translators

COMS W4115



Pieter Bruegel, *The Tower of Babel*, 1563

Prof. Stephen A. Edwards  
Fall 2007

Columbia University  
Department of Computer Science

## 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

## Instructor

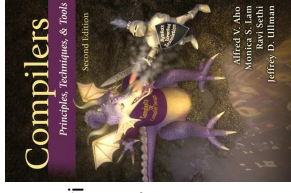
Prof. Stephen A. Edwards  
sedwards@cs.columbia.edu  
<http://www1.cs.columbia.edu/~sedwards/>  
462 Computer Science Building  
Office Hours: 4–5 PM Tuesday, 4–5 PM Wednesday

## Schedule

Tuesdays and Thursdays, 2:40 PM to 3:55 PM  
535 Mudd  
Lectures: September 4 to December 6  
Midterm: October 25  
Final: December 6 (in-class)  
Final project report: December 18  
Holidays: November 6 (Election day), November 22 (Thanksgiving)

## Required Text

Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman.  
*Compilers: Principles, Techniques, and Tools*.  
Addison-Wesley, 2006. Second Edition.



## Assignments and Grading

40% Programming Project  
20% Midterm (near middle of term)  
30% Final (at end of term)  
10% Individual homework

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

## Prerequisite: Java Fluency

You and your group will write perhaps 5000 lines of Java; you will not have time to learn it.

We will be using a tool that generates fairly complicated Java and it will be necessary to understand the output.

## Prerequisite: COMS W3157 Advanced Programming

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

## Prerequisite: 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://www1.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.  
Exception: CVN students do the project by themselves.  
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: If you're dumb enough to cheat, I'm smart enough to catch you.

# 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 language.

Suggested division of labor: Front-end, back-end, testing, documentation.

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

Exception: CVN students do the project by themselves.

## 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 Language*; see the class website).



## 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 25 soon  
Reference Manual October 18  
Final Report December 18

## Design a language?

A small, domain-specific language.  
Think of awk or php, not Java or C++.  
**Examples from earlier terms:**  
Quantum computing language  
Geometric figure drawing language  
Projectile motion simulation language  
Matlab-like array manipulation language  
Screenplay animation language

## Other language ideas

Simple animation language  
Model train simulation language  
Escher-like pattern generator  
Music manipulation language (harmony)  
Web surfing language  
Mathematical function manipulator  
Simple scripting language (à la Tcl)  
Petri net simulation language

# What's in a Language?

## 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++)
        a = b;
        b = c;
    return b;
}
```



## 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:

```
expr → expr + expr
      | expr - expr
      | expr * expr
      | expr / expr
      | digit
      | (expr)
```

## Semantics

Something may be syntactically correct but semantically nonsensical.

The rock jumped through the hairy planet.

Or ambiguous

The chickens are ready for eating.

## 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 beyond the scope of this class, but 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.

## FORTAN

**Before** **After: Expressions, control-flow**

```

gdc: pushl %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
.L9: leave
ret
.L5: subl %eax, %edx
jmp .L2

```

## APL

**Powerful operators, interactive language**

```

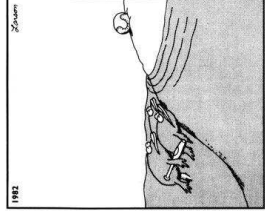
[0] Z←GAUSSIAN N←B←P←Q←R
[1] ⍲ Returns n random numbers having a Gaussian normal distribution
[2] A (with mean 0 and variance 1) Uses the Box-Muller method.
[3] ⍲ See Numerical Recipes in C, Pg. 289.
[4] P←10
[5] M←1+2*31 ⍲ largest integer
[6] L1←0+N÷P2 ⍲ a how many more we need
[7] L1←0+N÷P2 ⍲ a quit if none
[8] Q←1.3*G2 ⍲ approx num points needed
[9] P←1+(5*M-1)÷1+P*(4.2)*M ⍲ random points in -1 to 1 square
[10] B←(B0)⍲K1 ⍲ a distance from origin squared
[11] R←R/R ⍲ P←B/P ⍲ points within unit circle
[12] P←(-2*(P-R))*K.5
[13] Z←P, P←F.[1.5]F
[14] L2←Z⍲Z
[15] L2←Z⍲Z
[16] ⍲ Archbates, 12/16/1997 16:20:23.170

```

Source: Jim Weigang, <http://www.chilton.com/~jimw/gstrand.html>

## Great Moments in

## Programming Language Evolution



Great moments in evolution



From [callepress.com](http://callepress.com)

## 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).

```

## Assembly

**Before: numbers** **After: Symbols**

```

55 gcd: pushl %ebp
89F5 %esp, %ebp
8E4508 movl 8(%ebp), %eax
8B550C movl 12(%ebp), %edx
39D0 cmpl %edx, %eax
740D je .L9
39D0 .L7: cmpl %edx, %eax
7B08 jle .L5
29D0 subl %edx, %eax
39D0 .L2: cmpl %edx, %eax
75F6 jne .L7
C9 .L9: leave
C3 ret
29C2 .L5: subl %eax, %edx
EBF6 jmp .L2

```

## 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-doinvert)
          (search-backward "(")
          (forward-line -2)
          (while (or (looking-at "```")
                    (looking-at ".*")
                    (looking-at "\\s*")
                    (looking-at "```"))
            (forward-line 1)

```

## SNOBOL, Icon

**String-processing languages**

```

LETTER = 'ABCDEFGHIJKLMNQRSTUWXYZ$#%'
SP.CH = "+*~='\"/;< .> , - _"
SCOTA = SP.CH
SCOTA ',&' =
Q = ""
QUIT = Q FENCE BREAK(Q) Q
ELEM = ' ' Q ANY(SCOTA) | BREAK(SCOTA) | REM
F3 = (SPAN(' ') | REOS(0)) FENCE
F1 = BREAK(' ') | REM
CAOP = ('LCL' | 'SET') ANY('ABC') |
+ 'AIF' | 'AGO' | 'ACTR' | 'ANOP'
ATTR = ANY('TLSIEN')
ELEM3 = (/ FENCE *F3C ), | ATTR Q | ELEM
ASK360 = ASKNO FENCE
+ ( CAOP . OPERATION B F3C . OPERAND |
+ F2 . OPERATION B F3 . OPERAND |
+ B
+ REM . COMMENT
+ F1

```

Algol-68, source <http://www.ccsse.monash.edu.au/~lloyd/ideProgLang/Algol68/reenmerge.a68>

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

## BASIC

### Programming for the masses

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

## 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

### Purer functional language

```
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
```

## 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

## VisiCalc, Lotus 1-2-3, Excel

### The spreadsheet style of programming

	A	B
1	Hours	23
2	Wage per hour	\$ 5.36
3		
4	Total Pay	= B1 * B2

## sh, awk, perl, tcl, python

### 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
}
```

## Prolog

### Logic Language

```
edge(a, b). edge(b, c).
edge(c, d). edge(d, e).
edge(b, e). edge(d, f).
path(X, X).
path(X, Y) :-
  edge(X, Z), path(Z, Y).
```