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

*Compilers: Principles, Techniques, and Tools.*


Bug Al 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

- 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

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.


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

Each team will develop its own language.

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 Language*; see the class website).
Final Report Sections

1. Introduction: the proposal
2. Language Tutorial
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 lâ 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?
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.

```java
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:

\[
\begin{align*}
expr & \rightarrow \ expr + \ expr \\
| & \ expr - \ expr \\
| & \ expr \ast \ expr \\
| & \ expr \div \ expr \\
| & \ digit \\
| & \ (expr)
\end{align*}
\]
Components of a language: Semantics

What a well-formed program “means.”

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

‘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:

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

Ambiguous in Java:

```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
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
10  if (a .LT. b) then
10     a = a - b
10  else
10     b = b - a
10  endif
10  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).
(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)))
Powerful operators, interactive language, custom character set

```
[0] Z+GAUSSRAND N;B;F;M;P;Q;R
[1] \( \sim \) Returns \( \omega \) random numbers having a Gaussian normal distribution
[2] \( \sim \) (with mean 0 and variance 1) Uses the Box–Muller method.
[4] \( \sim \)
[5] Z+10
[6] M+\(-1+2\times31\) \( \sim \) largest integer
[7] L1:Q+N-\(\sim\)Z \( \sim \) how many more we need
[8] \( \sim \) (Q\leq0)/L2 \( \sim \) quit if none
[9] Q+[1.3\times Q\div2] \( \sim \) approx num points needed
[10] P+=\(-1+(2\div M-1)\times-1+?(Q,2)\times M \) \( \sim \) a random points in -1 to 1 square
[11] R+\(\sim\)/P\times P \( \sim \) a distance from origin squared
[12] B+\(\times R\neq 0\)\(\times R\rangle 1\) \( \sim \) distance from origin squared
[13] R+\(\times B\div R\) \( \sim \) points within unit
[14] F+\(-2\times(\Phi R)\times R\rangle 0.5\) \( \sim \) points within unit
[15] Z+Z,\times F,[1.5]F \( \sim \) ArchDate: 12/16/1997 16:20:23.170
[16] \( \sim \) L1
[17] L2:Z+N+Z
[18] \( \sim \)


At right: Datamedia APL Keyboard
```
Algol, Pascal, Clu, Modula, Ada

*Imperative, block-structured language, formal syntax definition, structured programming*

```plaintext
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

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.
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

```c
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

```haskell
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

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());
```
SQL T-Shirt

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

From thinkgeek.com
Prolog

Logic Language

\[
\begin{align*}
\text{witch}(X) & \iff \text{burns}(X) \land \text{female}(X). \\
\text{burns}(X) & \iff \text{wooden}(X). \\
\text{wooden}(X) & \iff \text{floats}(X). \\
\text{floats}(X) & \iff \text{sameweight}(\text{duck}, X). \\
\text{female}(\text{girl}). & \quad \{\text{by observation}\} \\
\text{sameweight}(\text{duck}, \text{girl}). & \quad \{\text{by experiment }\} \\
? \text{witch}(\text{girl}). & \quad \{\text{by experiment}\}
\end{align*}
\]