Programming Languages and Translators
COMS W4115
Prof. Stephen A. Edwards
Spring 2002
Columbia University
Department of Computer Science

Instructor
Prof. Stephen A. Edwards
sedwards@cs.columbia.edu
http://www.cs.columbia.edu/~sedwards/
462 Computer Science Building
Office Hours: 4–5 PM Monday, Wednesday

Schedule
Mondays and Wednesdays 2:40 to 3:55 PM
Room 207, Mathematics
January 23 to May 6
Midterm 1: March 13
Spring Break: March 18 and 20
Midterm 2: May 1

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
Michael L. Scott.
Programming Language Pragmatics
Available from Papyrus, 114th and Broadway.

Other Text
Andrew W. Appel.
Modern Compiler Implementation in Java.
Describes the Tiger language, which we are compiling.
Focuses more on compilers, less on languages.

Assignments and Grading
40% Programming Project
25% Midterm 1 (near middle of term)
25% Midterm 2 (at end of term)
10% Individual homework

Prerequisite: COMS W3156
Software Engineering
Teams will build a large software system.
Makefiles and possibly version control
Testing will be as important as development.

Prerequisite: COMS W3261
Computability
You need to understand grammars.
We will be working with regular and context-free languages.
Prerequisite: COMS W3824 Computer Organization
You need to be able to program in MIPS assembly language.
Your compiler will generate MIPS assembly code.

Class Website
Contains syllabus, lecture notes, and assignments.
Schedule will be continually updated during the semester.

Programming Project
Implement a compiler for the Tiger language.
Tiger is described in Appel, but we will deviate from the assignments there.
Compiler implemented in Java.
Some code generated by ANTLR.
Generates MIPS assembly code.

Programming assignments
1. Lexer, parser, and AST generator
2. Perform type checking (semantic analysis)
3. Translate into three-address code
4. Generate MIPS assembly
Two weeks each.

Teams
Immediately, start thinking about forming 4-5 person teams to do the programming project.
Each team will build its own compiler.
Think carefully about how you will divide the work.
Testing is as important as coding.

Late Policy
Very simple:
If you turn anything in late without the instructor's permission, you get no credit.

Collaboration
Collaborate with your team on the programming assignment.
Teams may share ideas, but not code. If I find two teams submitting similar files, both teams will receive zero credit, may flunk the class, and I may involve the dean.
Homework is to be done by yourself.

Topics
Syntax, Parsing, and ASTs
Names, Types, and Scopes
Control-flow and subroutines
Code generation
Functional and logic programming
Concurrency
Scripting languages

Types of Programming Languages
The world does not revolve around Java.
Imperative languages
Object-oriented languages
Functional languages
Logic languages
Dataflow languages
Imperative (von Neumann) Languages

What you are familiar with. e.g., C

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

Computation is the sequential modification of variables.
Programs are sequences of steps that evolve state.
Everything interesting has a side effect.

Again: if (a == b) goto Done

if (a < b) goto ALess
a = a - b;
goto Again

ALess: b = b - a;
goto Again

Done:

Object-Oriented Languages

Refinement of the imperative approach.
Memory partitioned into objects (small regions).
Objects have methods: imperative procedures to query or modify object state.

More disciplined than simple imperative languages.
Naturally enforce information hiding.
Currently taking over the world.

Object-Oriented Language: Smalltalk

```smalltalk
class name Polygon
    superclass Object
    instance variable names OurPen
        numSides
    new    "Create an instance"
        ^ super new getPen
    getPen "Get a pen for drawing polygons"
        OurPen <- Pen new defaultNib: 2
    draw "Draw a polygon"
        numSides timesRepeat: [OurPen go: sideLength;
            turn: 360 // numSides]
```

Functional Languages

Computation based on recursive definition of functions.
Function: produce a consistent result based exclusively on their arguments.
No side-effects. Mathematically very clean.
Declarative: program defines what the function is, not how to evaluate it.
Allows certain optimizations (e.g., laziness).
Do people think this way?

Functional Language: ML

A list of the form \((m, m+1, \ldots, n)\):

```ml
fun op through (m,n) =  
    if m > n  
        then nil  
        else m :: (m+1 through n)
```

Calculating area:

```ml
let pi = 3.14159;  
in
    pi * radius * radius
end;
```

Functional Languages

Virtually every assembly language
FORTRAN
C
Pascal
Modula-2
Algol
BASIC

Object-Oriented Languages

Simula 67
C++
Java
Modula-3

Functional Languages

LISP
ML
Haskell
Erlang
Logic Languages
Computation based on propositional logic.
You declare things; program execution is an attempt to satisfy what you declare.
Goal-directed search.
Interesting for AI-type applications.

Logic Languages: Prolog
rainy(seattle). % clause
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), cold(X). % implication

?- snowy(C). % query
C = rochester % response

Dataflow Languages
Computation based on exchange of data tokens among concurrently-running processes.
Nice match for engineering diagrams.
Fundamentally concurrent.
Awkward for decisions, modes, etc.

Dataflow Language: Lustre
Module counts edges: 0 → 1 transitions on the c input.
System is a collection of expressions evaluated in lockstep. Order from data dependencies.

node EDGECOUNT(c : bool)
returns (count : int)
let
   edge = false -> (c and not pre(c));
   edgcount = 0 ->
      if edge then pre(edgecount) + 1
      else pre(edgecount);
   tel