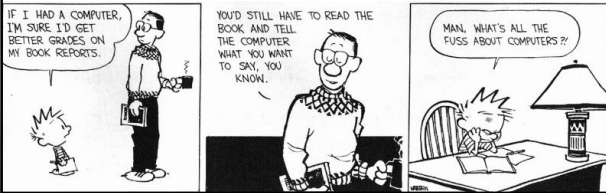


# Programming Language Design and Implementation

Wes Weimer  
TR 9:30-10:45  
MEC 214



---

---

---

---

---

---

---

---

## Cunning Plan

- Who Are We?
  - Wes, Pieter, Isabelle
- Administrivia
- What Is This Class About?
- Brief History Lesson
- Understanding a Program in Stages

#2

---

---

---

---

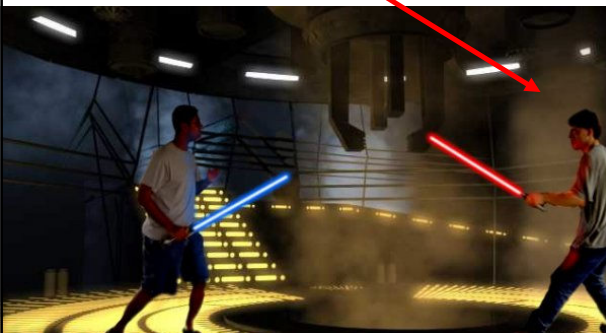
---

---

---

---

## Your Host For The Semester



---

---

---

---

---

---

---

---

## Course Staff - TA

- Pieter Hooimeijer
- Email: **ph4u**
- Isabelle Stanton
- **ils9d**



---

---

---

---

---

---

---

---



---

---

---

---

---

---

---

---

## Course Home Page

- google: **virginia cs 415**
- [www.cs.virginia.edu/~cs415/](http://www.cs.virginia.edu/~cs415/)
- Lectures slides are available before class
  - **You should still take notes!**
- Assignments are listed
  - also grading breakdown, regrade policies, etc.
- **Use the class forum for all public questions**

#6

---

---

---

---

---

---

---

---

## Discussion Sections

- There will be two one-hour-long “structured office hours” each week
  - Hosted by the TAs
- We will not take attendance, but you are encouraged to show up to one each week
  - Notes posted on web
  - For your benefit!
- Answer questions, go over lecture material, help and hints on the homework and projects
- Fill out the time signup sheet

47

---

---

---

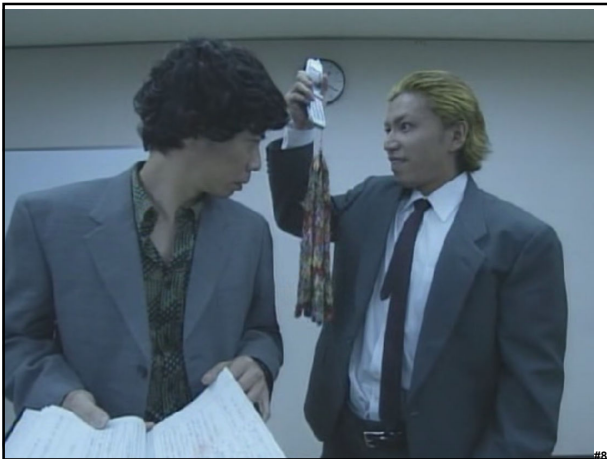
---

---

---

---

---



48

---

---

---

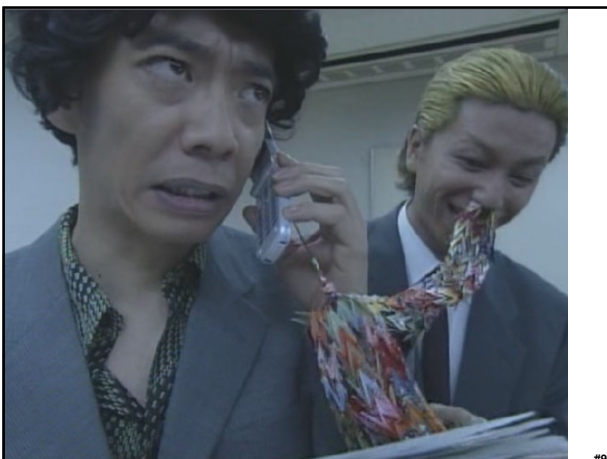
---

---

---

---

---



49

---

---

---

---

---

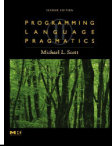
---

---

---

## Course Structure

- Course has **theoretical** and **practical** aspects
  - Best of both worlds!
- Need both in programming languages!
- **Reading = both**
  - Many external and optional readings
- **Written assignments = theory**
  - Class hand-in, right before lecture, pass/fail
- **Programming assignments = practice**
  - Electronic hand-in
- Strict deadlines



---

---

---

---

---

---

---

---

## Academic Honesty

- Don't use work from uncited sources
  - Including old code
- We use plagiarism detection software



#11

---

---

---

---

---

---

---

---

## The Course Project

- **A big project: an Interpreter!**
- ... in five easy parts
- Start early!



© Scott Adams, Inc./Dist. by UFS, Inc.

---

---

---

---

---

---

---

---

## How are Languages Implemented?

- Two major strategies:
  - [Interpreters](#) (take source code and run it)
  - [Compilers](#) (translate source code, run result)
  - Distinctions blurring (e.g., just-in-time compiler)
- Interpreters run programs “as is”
  - Little or no preprocessing
- Compilers do extensive preprocessing
  - Most implementations use compilers

#13

---

---

---

---

---

---

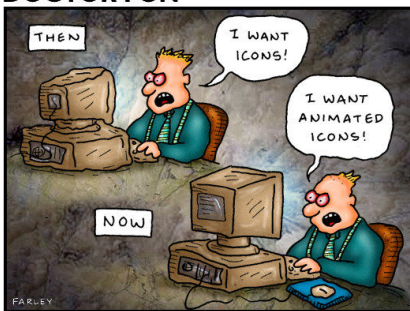
---

---

## Don't We Already Have Compilers?

DOCTOR FUN

19 Mar 97



Copyright © 1997 David Farley, d.farley@tercent.com  
<http://sunsite.unc.edu/Dave/drfun.html>  
This cartoon is made available on the Internet for personal viewing only.  
Opinions expressed herein are solely those of the author.

Progress

#14

---

---

---

---

---

---

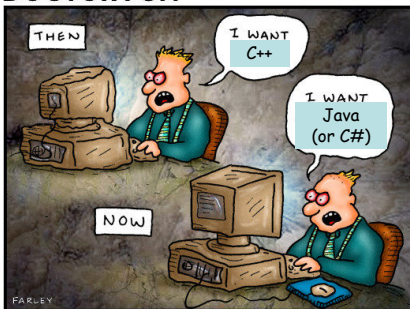
---

---

## Dismal View Of Prog Languages

DOCTOR FUN

19 Mar 97



Copyright © 1997 David Farley, d.farley@tercent.com  
<http://sunsite.unc.edu/Dave/drfun.html>  
This cartoon is made available on the Internet for personal viewing only.  
Opinions expressed herein are solely those of the author.

Progress

#15

---

---

---

---

---

---

---

---

## (Short) History of High-Level Languages

- 1953 IBM develops the 701 “Defense Calculator”
  - 1952, US formally ends occupation of Japan
  - 1954, Brown v. Board of Education of Topeka, Kansas
- All programming done in assembly
- Problem: Software costs exceeded hardware costs!
- John Backus: “Speedcoding”
  - An interpreter
  - Ran 10-20 times slower than hand-written assembly

---

---

---

---

---

---

---

---

## FORTRAN I

- 1954 IBM develops the 704
- John Backus
  - Idea: translate high-level code to assembly
  - Many thought this impossible
- 1954-7 FORTRAN I project
- By 1958, >50% of all software is in FORTRAN
- Cut development time dramatically
  - (2 weeks → 2 hours)

#17

---

---

---

---

---

---

---

---

## FORTRAN I

- The first compiler
  - Produced code almost as good as hand-written
  - Huge impact on computer science
- Led to an enormous body of theoretical work
- Modern compilers preserve the outlines of FORTRAN I

#18

---

---

---

---

---

---

---

---

### Q: TV (100 / 842)

- In this 1985-1992 ABC television series, the gunless title character Angus works for Pete and the Phoenix Foundation and makes heavy use of his Swiss Army knife and duct tape.

---

---

---

---

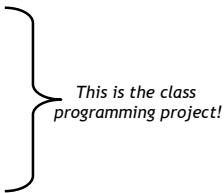
---

---

---

---

### The Structure of an Interpreter

1. Lexical Analysis
  2. Parsing
  3. Semantic Analysis
  4. Optimization (optional)
  5. Run It!
- 

The first 3, at least, can be understood by analogy to how humans comprehend English.

#20

---

---

---

---

---

---

---

---

### Lexical Analysis

- First step: recognize words.
  - Smallest unit above letters

**This is a sentence.**

- Note the
  - Capital "T" (start of sentence symbol)
  - Blank " " (word separator)
  - Period "." (end of sentence symbol)

#21

---

---

---

---

---

---

---

---

## More Lexical Analysis

- Lexical analysis is not trivial. Consider:  
**How d'you break "this" up?**
- Plus, programming languages are typically more cryptic than English:

**\*p->f += -.12345e-6**



#22

---

---

---

---

---

---

---

---

## And More Lexical Analysis

- **Lexical analyzer** divides program text into "words" or **tokens**

**if x == y then z = 1; else z = 2;**

- Broken up:

**if, x, ==, y, then, z, =, 1, ;, else, z, =, 2, ;**

#23

---

---

---

---

---

---

---

---

## Parsing

- Once words are understood, the next step is to understand sentence structure

- **Parsing** = Diagramming Sentences

- The diagram is a tree

#24

---

---

---

---

---

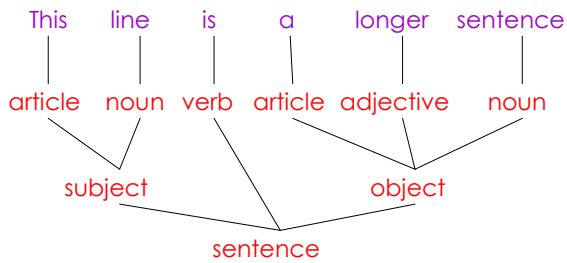
---

---

---



## Diagramming a Sentence



#25

---

---

---

---

---

---

---

---

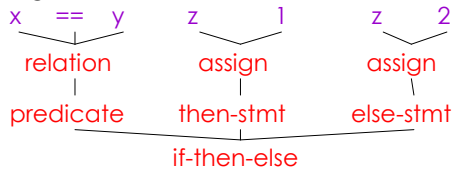
## Parsing Programs

- Parsing program expressions is the same

- Consider:

if x == y then z = 1; else z = 2;

- Diagrammed:



#26

---

---

---

---

---

---

---

---

## Semantic Analysis

- Once sentence structure is understood, we can try to understand "meaning"
  - But meaning is **too hard for compilers**
- Compilers perform limited analysis to catch inconsistencies: **reject bad programs early!**
- Some do more analysis to improve the performance of the program

#27

---

---

---

---

---

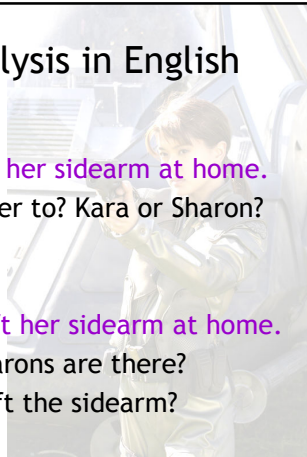
---

---

---

## Semantic Analysis in English

- Example:  
Kara said Sharon left her sidearm at home.  
What does “her” refer to? Kara or Sharon?
- Even worse:  
Sharon said Sharon left her sidearm at home.  
How many Sharons are there?  
Which one left the sidearm?



---

---

---

---

---

---

---

---

## Semantic Analysis in Programming

- Programming languages define strict rules to avoid such ambiguities

- This C++ code prints “4”; the inner definition is used

```
{  
  int Sydney = 3;  
  {  
    int Sydney = 4;  
    cout << Sydney;  
  }  
}
```

Scoping or aliasing problem.

#29

---

---

---

---

---

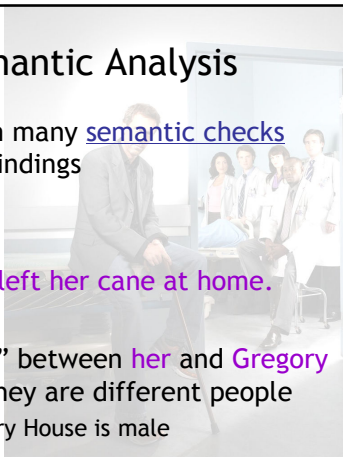
---

---

---

## More Semantic Analysis

- Compilers perform many semantic checks besides variable bindings
- Example:  
Gregory House left her cane at home.
- A “type mismatch” between her and Gregory House; we know they are different people
  - Presumably Gregory House is male



---

---

---

---

---

---

---

---

## Optimization

- No strong counterpart in English, but akin to editing
- **Automatically modify programs** so that they
  - Run faster
  - Use less memory
  - In general, conserve some resource
- The project has no optimization component

#31

---

---

---

---

---

---

---

---

## Code Generation

- Produces assembly code (usually)
  - which is then assembled into executables by an assembler
- A translation into another language
  - Analogous to human translation
- We will **not do codegen in this class**
  - Instead you will interpret the program directly!

#32

---

---

---

---

---

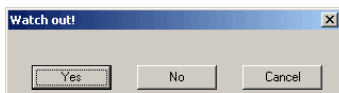
---

---

---

## Issues

- Compiling and interpreting are almost this simple, but there are **many pitfalls**.
- Example: How are erroneous programs handled?
- Language design has big impact on compiler
  - Determines what is easy and hard to compile
  - Course theme: **trade-offs in language design**



#33

---

---

---

---

---

---

---

---

## Languages Today

- The overall structure of almost every compiler & interpreter follows our outline
- The proportions have changed since FORTRAN
  - Early: lexing, parsing most complex, expensive
  - Today: optimization dominates all other phases, lexing and parsing are cheap

#34

---

---

---

---

---

---

---

---

## Trends in Languages

- Optimization for speed is less interesting. But:
  - scientific programs
  - advanced processors (Digital Signal Processors, advanced speculative architectures)
  - Small devices where speed = longer battery life
- Ideas we'll discuss are used for improving code reliability:
  - memory safety
  - detecting concurrency errors (data races)
  - type safety
  - automatic memory management
  - ...

#35

---

---

---

---

---

---

---

---

## Why Study Prog. Languages?

- Increase capacity of expression
  - See what is possible
- Improve understanding of program behavior
  - Know how things work "under the hood"
- Increase ability to learn new languages
- Learn to build a large and reliable system
- See many basic CS concepts at work

#36

---

---

---

---

---

---

---

---

## What Will You Do In This Class?

- **Reading** (textbook, outside sources)
- **Learn** about different kinds of languages
  - Imperative vs. Functional vs. Object-Oriented
  - Static typing vs. Dynamic typing
  - etc.
- **Learn to program** in different languages
  - Python, Ruby, ML, "Cool" (= micro-Java)
- **Write an interpreter!**

#37

---

---

---

---

---

---

---

---

## What Is This?

A lungo il mio cuore di tali ricordi ha voluto colmarsi!	Długo, długo moje serce przepelnione było takimi wspomnieniami!
Come un vaso in cui le rose sono state dissetate:	Był jak waza, w której kiedyś róże destylowały:
Puoi romperlo, puoi distruggere il vaso se lo vuoi,	Możesz sprawić by pękła, możesz gruchotać wazę jeśli chcesz,
Ma il profumo delle rose sarà sempre tutt'intorno.	Ale zapach róż będzie wciąż czuć dookoła.
Mon coeur est brillant rempli de tels souvenirs	Lang, lang soll die Erinnerung in meinem Herzen klingen!
Comme un vase dans lequel des roses ont été distillées:	Gleich einer Vase, drin Rosen sich einst tränkten:
Tu peux le briser, tu peux détruire le vase si tu le désires.	Lass sie zerbrechen, lass sie zerspringen,
Mais la senteur des roses sera toujours là.	Der Duft der Rose bleibt immer hängen.
Muito, muito tempo seja meu coração preenchido com tais lembranças!	
Tal qual o vaso onde rosas foram uma vez destiladas:	
Pode quebrar, pode estilhaçar o vaso se o desejas,	
Mas perdurará para sempre o aroma das rosas perfumadas.	

#38

---

---

---

---

---

---

---

---

## The Rosetta Stone

- The first programming assignment involves **writing the same simple (50-75 line) program in five languages**
  - Ruby, Python, OCaml, Cool and C
- PA0, due next Friday, requires you to write the program in just one language (you pick)
- PA1, due one week later, requires all five

Long, long be my heart with such memories fill'd!  
Like the vase in which roses have once been distill'd:  
You may break, you may shatter the vase if you will,  
But the scent of the roses will hang round it still.  
- Thomas Moore (Irish poet, 1779-1852)

#39

---

---

---

---

---

---

---

---

## Homework

- Scott Book, parts of Chapter 10 (for Tuesday)
- Get started on PA0 (due in 8 days)

## Questions?



---

---

---

---

---

---

---

---