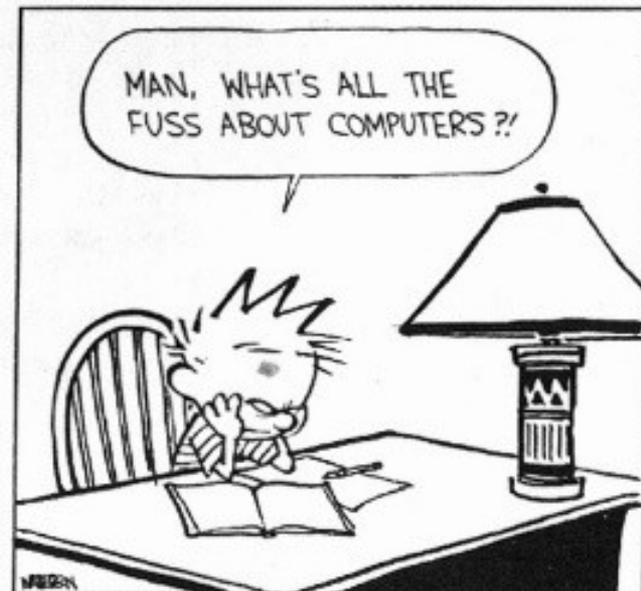
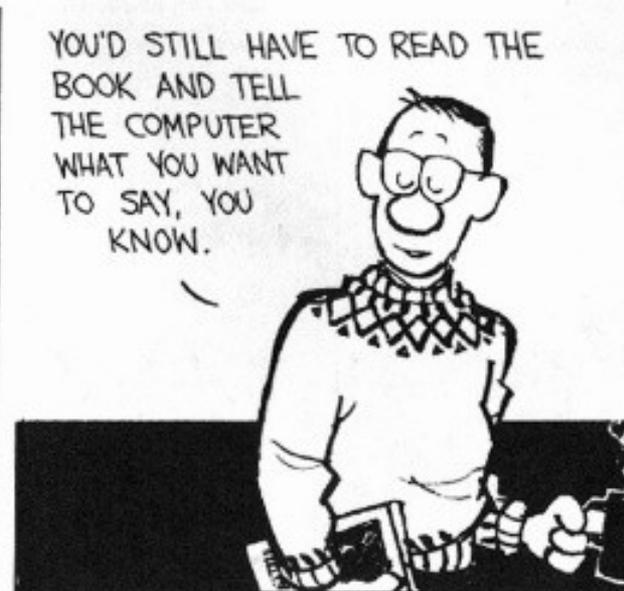
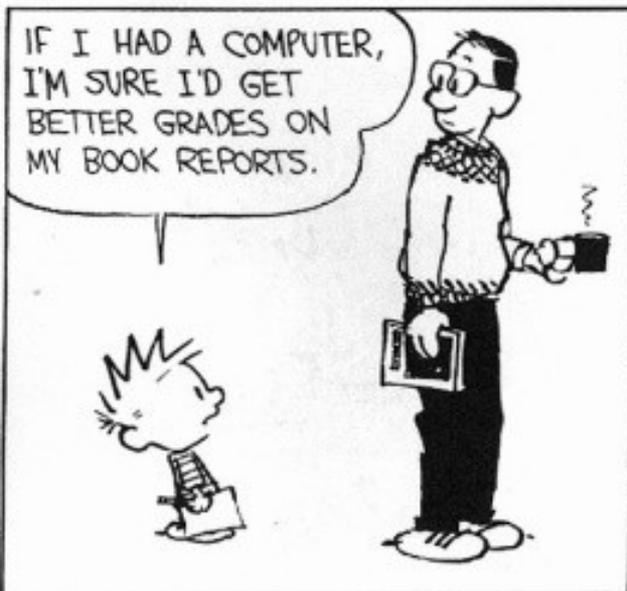


# Language Design and Implementation

Wes Weimer

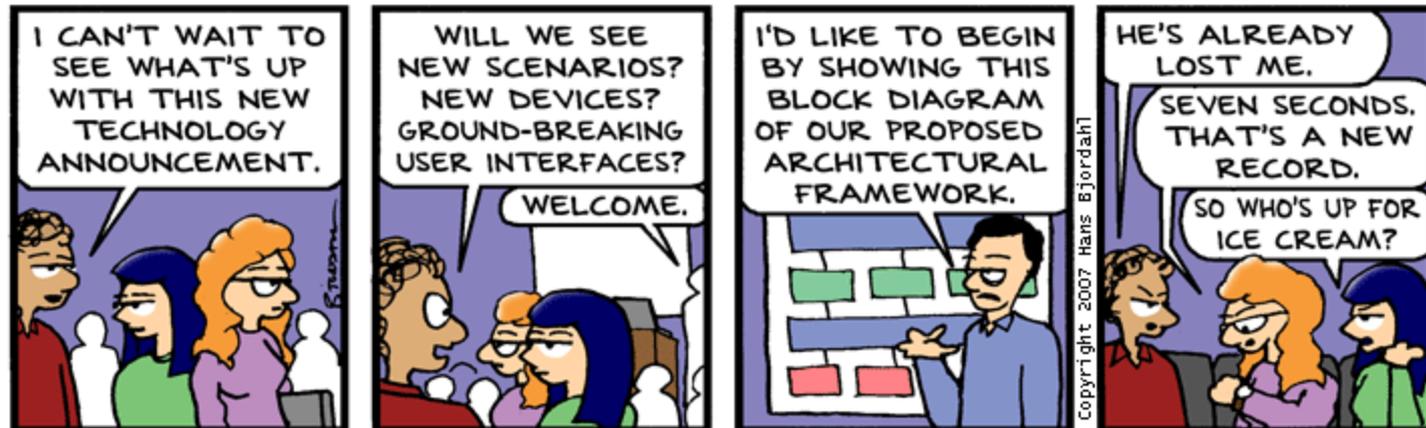
TR 3:30 - 4:45

Thornton E-303



# Cunning Plan

- Administrivia
  - Webpage, Wes, Kevin Leach, undergrad TAs
- What Is This Class About?
- Easy or Hard? Work and Grading.
- Understanding a Program in Stages



# Course Home Page

- Find via Weimer webpage or Lou's List
- <http://dijkstra.cs.virginia.edu/ldi/>
- 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

# Language Design and Implementation Course Goals

- At the end of this course, you will be acquainted with the fundamental concepts in the **design and implementation** of high-level programming **languages**. In particular, you will understand the **theory and practice** of **lexing, parsing, semantic analysis, and code interpretation**. You will also have gained practical experience programming in **different languages**.

# Who Cares?

- In most cases, there is a clear mapping between 4000-level electives and jobs or internships:
- Take Databases (4750) -> work at Oracle
- Take E-Commerce (4753) -> work at Amazon
- Take Networks (4457) -> work at Cisco
- Take Graphics (4810) -> work at Nvidia
- Take LDI (4501) -> ???
- *Which companies develop compilers or interpreters?*

# Microsoft

- Visual Studio, Excel, etc.

The screenshot shows the Visual Studio website's 'News' section. The header includes the Visual Studio logo, navigation links (Products, Features, Downloads, News, Support, Marketplace, Documentation), and links for MSDN Subscriptions, Sign in, and a 'Free Visual Studio' button. The main content area has a purple background with the heading 'Visual Studio Team Services Jobs'. Below the heading is the text 'Become a member of the Visual Studio Team Services Team' followed by three bullet points: 'Work with the latest technologies', 'Work on a fast-paced agile team', and 'Have a big impact on the software industry through building an innovative service'. At the bottom right of this section are three buttons: 'Send Resume', 'Open Positions', and 'Life at Microsoft'. Below the purple section is a white section with the heading 'Join us!' and a paragraph of text: 'Become a key member of the Visual Studio Team Services (VSTS) service team led by Brian Harry and build the next generation of development services in the cloud! VSTS provides software development teams with version control, build automation, agile work management, social experiences and more to nearly 3,000,000 users.'

# Oracle

- Java Compiler, Java Virtual Machine

The screenshot shows the Oracle Java website. The Oracle logo is in the top left. The navigation bar includes links for Sign In/Register, Help, Country, Communities, I am a..., and I want to..., along with a search box. Below the navigation bar are links for Products, Solutions, Support, Training, Partners, About, and OTN. The main content area features the text "Java Software" and "Create the Future". A dropdown menu is open, listing various roles: Java Developer, Database Administrator / Developer, System Admin / Developer, Architect, C-Level Executive (Chief Financial Officer, Chief Human Resources Officer, Chief Information Officer), and Other Roles (Analyst, Investor, Job Seeker, Partner, Student, Midsize Company). The "Job Seeker" link is highlighted. The background of the page features a man in a blue hoodie and a "20 YEARS 1995-2015" badge.

# Intel

- ICC



## Leadership Application Performance

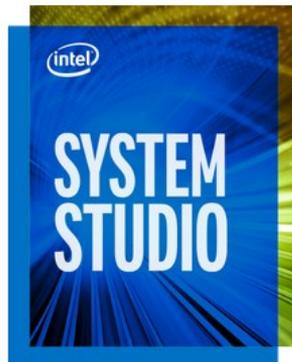
- Boost C++ application performance
- Future-proof code by making code that scales
- Plugs right into your development environment

If you are here, you are looking for ways to make your application run faster. Boost performance by augmenting your development process with the Intel® C++ Compiler. The Intel C++ Compiler plugs right into popular development environments like Visual Studio\*, Eclipse\*, XCode\*, and Android Studio\*; The Intel C++ Compiler is compatible with popular compilers including Visual C++\* (Windows\*) and GCC (Linux\*, OS X\* and Android\*).

The Intel C++ Compiler is available in four products based on your application development needs:



Intel® C++ Compiler in Intel® Parallel Studio XE



Intel® C++ Compiler in Intel® System Studio



Intel® C++ Compilers in Intel® INDE (support only)



Intel® Bi-Endian Compiler

# Google

- Go, Dart, etc.

The screenshot shows the Go Programming Language website. At the top, there is a navigation bar with links for 'GET STARTED', 'FUNDAMENTALS', 'WEB', 'SERVER', and 'MORE'. A search bar is located on the right. The main content area features a large blue banner with the text 'Scalable, productive' and 'Development'. Below this, there is a section titled 'The Go Programming Language' with a navigation bar containing buttons for 'Documents', 'Packages', 'The Project', 'Help', 'Blog', and a search bar. The 'Try Go' section is highlighted, showing a code editor with the following code:

```
// You can edit this code!  
// Click here and start typing.  
package main  
  
import "fmt"  
  
func main() {  
    fmt.Println("Hello, 世界")  
}
```

Below the code editor, there is a dropdown menu showing 'Hello, World!' and buttons for 'Run', 'Share', and 'Tour'. To the right of the 'Try Go' section, there is a text block: 'Go is an open source programming language that makes it easy to build simple, reliable, and efficient software.' Below this text is a cartoon illustration of a bear's face. At the bottom right, there is a button labeled 'Download Go' with the text: 'Binary distributions available for Linux, Mac OS X, Windows, and more.'

# Wind River, Green Hills

## • Embedded!

### DIAB COMPILER

For over 25 years, Wind River Diab Compiler has been helping industrial, medical, and aerospace industries. Diab Compiler has a tiny footprint, and produce high-quality, standards-compliant code.



Diab Compiler's unique optimization technology generates extremely fast, high-quality object code in the smallest possible footprint.



Due to collaboration with Wind River, Diab Compiler is now available on older processors of time to allow for the development of the compiler for



Leading the Embedded World



Products Markets Benefits Services Support Partners News About

### Jobs - Opportunities in the USA

Green Hills Software is always looking for qualified Engineering, Sales, and Marketing staff. Please submit your resume to the Corporate Office where it will be processed and reviewed by the hiring manager.

Click on a job title below for a complete description of the position:

- [Corporate Field Applications Engineer](#) (Santa Barbara, CA)
- [Embedded Software Consultant](#) (Santa Barbara, CA)
- [Embedded Solutions Test Engineer](#) (Santa Barbara, CA)
- [Field Engineer](#) (Santa Barbara, CA)
- [Field Services Engineer](#) (Santa Barbara, CA)
- [Functional Safety Software Engineer](#) (Santa Barbara, CA)
- [Product Engineer](#) (Santa Barbara, CA)
- [Sales Managers](#) (location TBD)
- [Software Development Engineer](#) (Santa Barbara, CA)
- [Technical Marketing Engineer](#) (Santa Barbara, CA)

**Click here for information on applying.**

Green Hills Software is an Equal Opportunity / Affirmative Action Employer.



### Software Development Engineer (Santa Barbara, CA)

#### Job description:

A software engineer has complete engineering responsibility for one or more major components of the Green Hills product line. For an experienced programmer this is a satisfying position in which you have personal responsibility for creating a tool used by thousands of programmers around the world. Our engineers are involved in Language Front Ends, Code Generators, Real Time Operating Systems, our MULTI Development Environment, our Secure Workstation, and Target Systems.

Here are the groups for which we are hiring:

- **Compiler:** Create, update, and maintain a language front end or a target architecture backend for the highly-optimizing family of Green Hills compilers. A compiler engineer might work on new language extensions, specific cutting-edge optimizations for the latest chips to hit the market, or on general optimizations that will benefit our entire product line. An ideal candidate understands low level microarchitecture designs and is comfortable working with assembly code, yet can also develop tools written in high level languages.

# Wait, what? Embedded?

- Curiosity Mars Rover, Cell Phones, Satellites, Engine Control Modules, Computed Radiology, Fighter Jets, Digital Cameras, Turbines, Anti-Lock Brakes, Wii U Game Console, ...



Eight years ago, NASA Jet Propulsion Laboratory (JPL) first began its work on the Mars Science Laboratory rover, Curiosity. Because of its long record of success with Wind River® on more than 20 JPL missions, NASA chose VxWorks® for the most technologically advanced autonomous robotic spacecraft and geologist set ever to be deployed by any space venture. Wind River VxWorks powered the

craft's controls from the second the rocket left Earth on November 26, 2011, to its successful landing in the Gale Crater on Mars on August 5, 2012, and will support Curiosity's exploratory capability throughout the life of the mission.

Stay tuned for future updates as Wind River VxWorks continues to play a strategic role in NASA's groundbreaking mission to determine whether Mars is or has ever been capable of supporting life and to assess the planet's habitability for future human missions.

# Adobe

- Photoshop contains interpreters ...

## 2 Photoshop Scripting Basics

This chapter provides an overview of scripting for Photoshop, describes scripting support for the scripting languages AppleScript, VBScript, and JavaScript, how to execute scripts, and covers the Photoshop object model. It provides a simple example of how to write your first Photoshop script.

If you are familiar with scripting or programming languages, you most likely will want to skip much of this chapter. Use the following list to locate information that is most relevant to you.

- For more information on the Photoshop object model, see [“Photoshop Object Model” on page 11](#).
- For information on selecting a scripting language, refer to the *Introduction to Scripting* guide.
- For examples of scripts created specifically for use with Photoshop, see Chapter 3, [“Scripting Photoshop” on page 21](#).
- For detailed information on Photoshop objects and commands, please use the reference information in the three reference manuals provided with this installation: *Adobe Photoshop CC 2015 AppleScript Scripting Reference*, *Adobe Photoshop CC 2015 Visual Basic Scripting Reference*, and *Adobe Photoshop CC 2015 JavaScript Scripting Reference*.

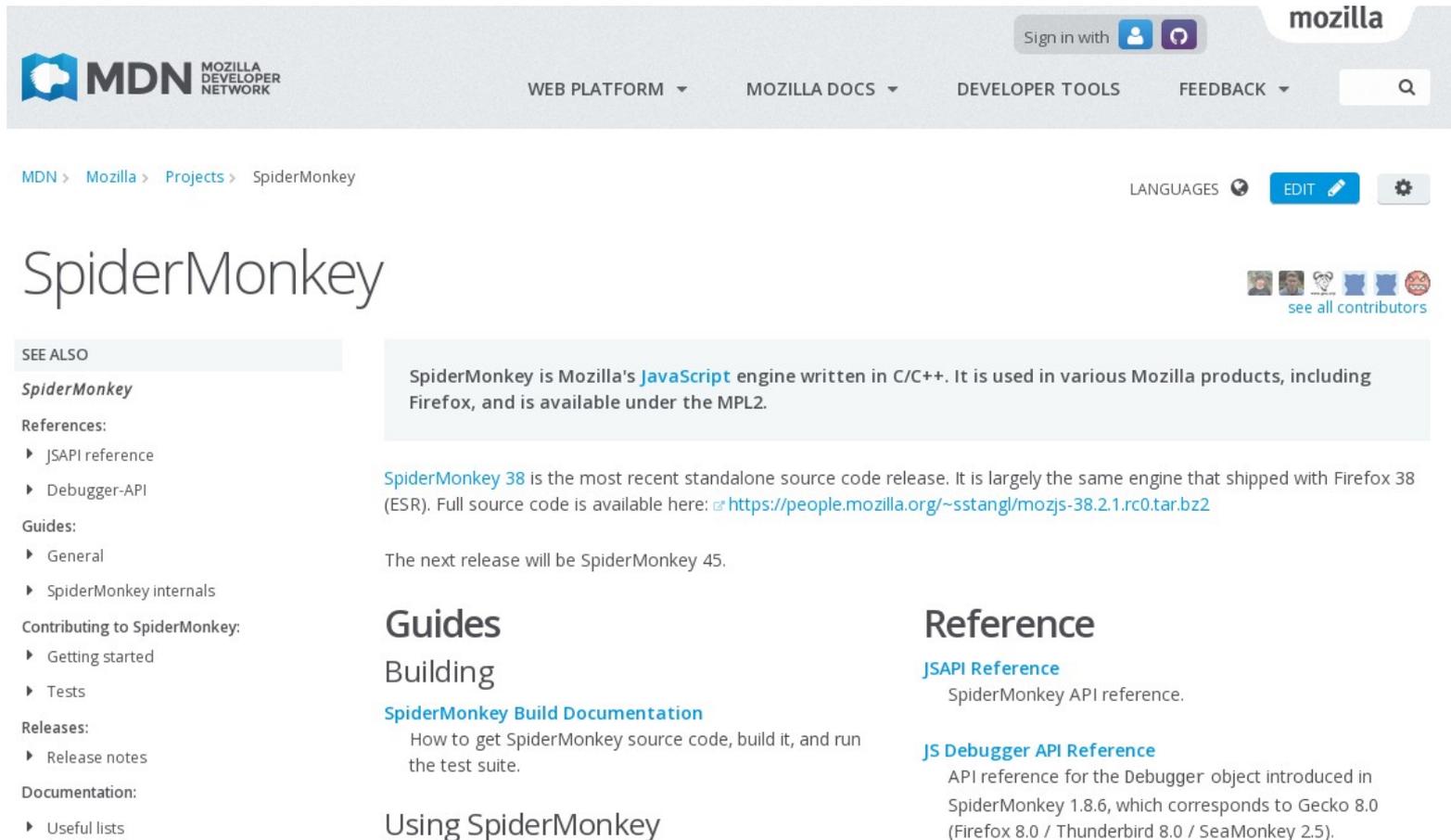
**NOTE:** You can also view information about the Photoshop objects and commands through the object browsers for each of the three scripting languages. See [“Viewing Photoshop Objects, Commands, and Methods” on page 21](#).

### Scripting Overview

A script is a series of commands that tells Photoshop to perform a set of specified actions, such as applying different filters to selections in an open document. These actions can be simple and affect only a single object, or they can be complex and affect many objects in a Photoshop document. The actions can call Photoshop alone or invoke other applications.

# Mozilla

- SpiderMonkey JavaScript Engine



The screenshot shows the MDN (Mozilla Developer Network) page for SpiderMonkey. The page header includes the MDN logo, navigation links for 'WEB PLATFORM', 'MOZILLA DOCS', 'DEVELOPER TOOLS', and 'FEEDBACK', and a search bar. The breadcrumb trail is 'MDN > Mozilla > Projects > SpiderMonkey'. The main heading is 'SpiderMonkey'. A 'SEE ALSO' section lists 'SpiderMonkey' with a sub-section for 'References' containing links to 'JSAPI reference' and 'Debugger-API', and a 'Guides' section with links to 'General' and 'SpiderMonkey internals'. A 'Contributing to SpiderMonkey' section lists 'Getting started' and 'Tests'. A 'Releases' section lists 'Release notes'. A 'Documentation' section lists 'Useful lists'. A light blue box contains the text: 'SpiderMonkey is Mozilla's JavaScript engine written in C/C++. It is used in various Mozilla products, including Firefox, and is available under the MPL2.' Below this, a paragraph states: 'SpiderMonkey 38 is the most recent standalone source code release. It is largely the same engine that shipped with Firefox 38 (ESR). Full source code is available here: https://people.mozilla.org/~sstangl/mozjs-38.2.1.rc0.tar.bz2'. Another paragraph says: 'The next release will be SpiderMonkey 45.' The page is divided into three columns: 'Guides' with a sub-section 'Building' containing a link to 'SpiderMonkey Build Documentation' and a paragraph 'How to get SpiderMonkey source code, build it, and run the test suite.', and 'Using SpiderMonkey'; 'Reference' with a sub-section 'JSAPI Reference' containing a paragraph 'SpiderMonkey API reference.', and a sub-section 'JS Debugger API Reference' containing a paragraph 'API reference for the Debugger object introduced in SpiderMonkey 1.8.6, which corresponds to Gecko 8.0 (Firefox 8.0 / Thunderbird 8.0 / SeaMonkey 2.5).'; and a 'see all contributors' link with a row of contributor avatars.

# Epic Games

- Unreal Engine: Blueprints Scripting

← Unreal Engine 4 Documentation

SHARE: [f](#) [t](#) [r](#) [in](#) [G](#)

## Blueprints Visual Scripting

Unreal Engine 4.9



## QuakeC

**QuakeC** is an [interpreted language](#) developed in 1996 by [id Software](#) to program parts of the [video game Quake](#). A programmer is able to customize *Quake* to great extent by adding new weapons, changing game logic and physics, and programming new scenarios. It can be used to control many aspects of the game, such as AI, triggers, or changes in the level. The only game engine to use QuakeC. Following engine modules for customization written in C and C++ from [id Tech 4](#) on.

The **Blueprints Visual Scripting** system in Unreal Engine is a complete gameplay scripting system based on the concept of using a node-based interface to create gameplay elements from within Unreal Editor. This system is extremely flexible and powerful as it provides the ability for designers to use virtually the full range of concepts and tools generally only available to programmers.

Through the use of Blueprints, designers can prototype, implement, or modify virtually any gameplay element, such as:

- Contents
- Overview
- Limitations

**Typing discipline** static, strong

### Major implementations

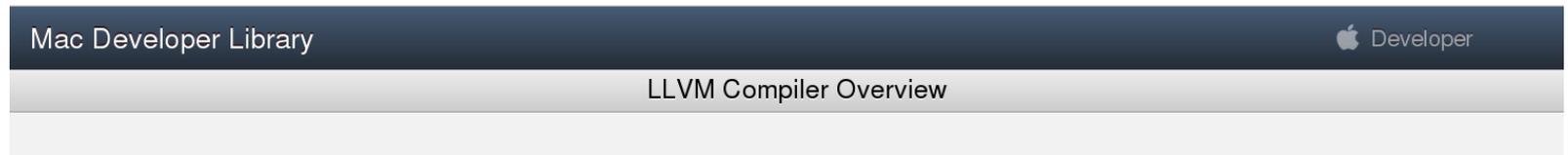
Quake C Compiler, FastQCC, QCCx, GMQCC

### Influenced by

C

# Apple

- Objective-C. LLVM.



## LLVM Compiler Overview

The LLVM compiler is the next-generation compiler, introduced in Xcode 3.2 for Snow Leopard, based on the open source [LLVM.org](http://LLVM.org) project. The LLVM.org project employs a unique approach of building compiler technologies as a set of libraries. Capable of working together or independently, these libraries enable rapid innovation and the ability to attack problems never before solved by compilers. Multiple technology groups within Apple are active contributors within the LLVM.org community, and they use LLVM technology to make Apple platforms faster and more secure.

In Xcode, the LLVM compiler uses the Clang front end (a C-based languages project on LLVM.org) to parse source code and turn it into an interim format. Then the LLVM code generation layer (back end) turns that interim format into final machine code. Xcode also includes the LLVM GCC compiler, which uses the GCC compiler front end for maximum compatibility, and the LLVM back end, which takes advantage of LLVM's advanced code generator. This shows the flexibility of a library-based approach to compiler development. There are many other features, such as link-time optimization, more detailed diagnostic information, and even static analysis, that are made available to Xcode due to the adoption of LLVM.

## About Objective-C

Objective-C is the primary programming language you use when writing software for OS X and iOS. It's a superset of the C programming language and provides object-oriented capabilities and a dynamic runtime. Objective-C inherits the syntax, primitive types, and flow control statements of C and adds syntax for defining classes and methods. It also adds language-level support for object graph management and object literals while providing dynamic typing and binding, deferring many responsibilities until runtime.

# Compilers and Interpreters

- Back End Optimization, Chips, etc.
  - Intel, AMD, nVidia, Green Hills, etc.
- Platform Vendors
  - Apple, Oracle, etc.
- Tooling, IDEs
  - Microsoft, Google, etc.
- Domain-Specific Languages
  - Photoshop, Game Studies, MATLAB, SQL, Wolfram Alpha, etc.

# Surprise: Flash, Postscript.

## ^ The language



PostScript is a Turing-complete programming language, belonging to the [concatenative](#) group. Typically, PostScript programs are not produced by humans, but by other programs. However, it is possible to write computer programs in PostScript just like any other programming language.<sup>[5]</sup>

## ActionScript



PostScript is an [interpreted](#), [stack-based](#) language similar to those found in [Lisp](#), [scoped memory](#) and, since language level [Polish notation](#), which makes the order of operations unambiguous. One has to keep the layout of the [stack](#) in mind. Most operations pop from the stack, and place their results onto the stack. [Literals](#) (for integers, strings, and booleans) are pushed onto the stack. Sophisticated data structures can be built on the system, which sees them all only as arrays and dictionaries, and "types" is left to the code that implements them.

ActionScript is an [object-oriented programming](#) language originally developed by [Macromedia Inc.](#) (now dissolved into [Adobe Systems](#)). It is a derivation of [HyperTalk](#), the scripting language for [HyperCard](#).<sup>[2]</sup> It is now a dialect of [ECMAScript](#) (meaning it is a superset of the syntax and semantics of the language more widely known as [JavaScript](#)), though it originally arose as a sibling, both being influenced by HyperTalk.

ActionScript is used primarily for the development of websites and software targeting the [Adobe Flash Player](#) platform, used on [Web pages](#) in the form of embedded SWF files.

ActionScript 3 is also used with [Adobe AIR](#) system for the development of desktop and mobile applications. The language itself is open-source in that its specification is offered free of charge<sup>[3]</sup> and both an open source compiler (as part of [Apache Flex](#)) and open source virtual machine ([Mozilla Tamarin](#)) are available.

ActionScript is also used with [Scaleform GfX](#) for the development of 3D video game user interfaces and HUDs.

### ActionScript



<b>Paradigm</b>	Multi-paradigm: object-oriented (prototype-based), functional, imperative, scripting
<b>Designed by</b>	Gary Grossman
<b>Developer</b>	Macromedia (now dissolved into Adobe Systems)
<b>First appeared</b>	1998
<b>Stable release</b>	3.0 / June 27, 2006
<b>Typing discipline</b>	strong, static
<b>Website</b>	<a href="#">adobe</a>
<b>Major implementations</b>	
Adobe Flash Player, Adobe AIR, Apache Flex, Scaleform GfX	
<b>Influenced by</b>	
<a href="#">JavaScript</a> , <a href="#">Java</a>	

# Surprise: Flash, Postscript.

## ^ The language



PostScript is a Turing-complete programming language, belonging to the [concatenative](#) group. Typically, PostScript programs are not produced by humans, but by other programs. However, it is possible to write computer programs in PostScript just like any other programming language.<sup>[5]</sup>

## ActionScript



PostScript is an [interpreted](#), [stack-based](#) language similar to

those found in [Lisp](#), [scoped memory](#) and, since language le

[Polish notation](#), wh

one has to keep the

the stack, and plac

on the stack. Sophi

system, which sees

"types" is left to the

ActionScript is an [object-oriented programming](#) language originally developed by [Macromedia Inc.](#) (now dissolved into [Adobe Systems](#)). It is a

ActionScript

Flash Player, Your Printer,  
Your Cell Phone, Acrobat Reader:  
they are contain *Interpreters*.

are available.

ActionScript is also used with [Scaleform GFx](#) for the development of 3D video game user interfaces and HUDs.

**Stable release** 3.0 / June 27, 2006

**Typing discipline** strong, static

**Website** [adobe](#)

### Major implementations

[Adobe Flash Player](#), [Adobe AIR](#), [Apache Flex](#), [Scaleform GFx](#)

### Influenced by

[JavaScript](#), [Java](#)

# Who Cares?

- The *computer* is unique among “machines” (e.g., lever, pulley, etc.) in that it magnifies our mental force rather than our physical force.
  - Computers can assist with decision making, model and predict outcomes, etc.
- **Programming Languages** are the mechanism for communicating with and commanding the only tool available that magnifies your mind.

# Plus Work, Double-Plus Easy

- Unhappiness is related to unrealized desires or unmet expectations
- CS 4610 is arguably the department's most difficult elective: 5 credits of work in a 3-credit course with a tough curve
- **Language Design and Implementation** is an approachable, streamlined variant
  - 3.5 credits of work in a 3-credit course with a very very generous curve

# In A Nutshell

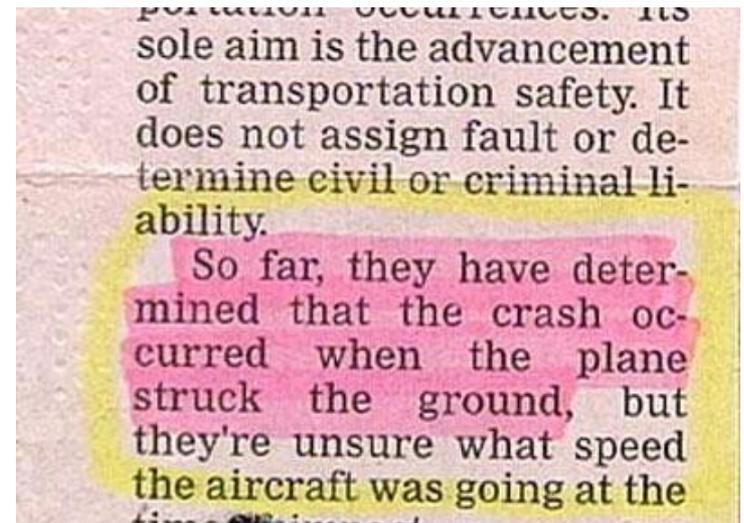
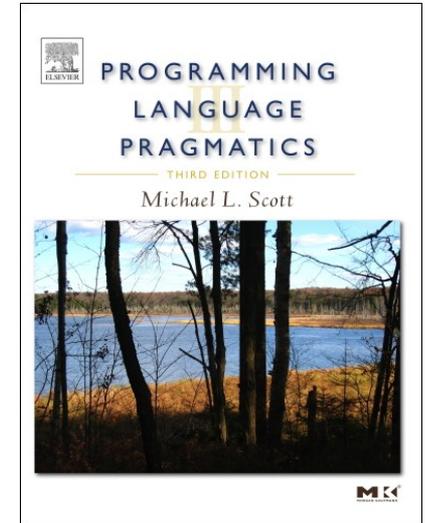
- Language Design and Implementation is
  - One notch more work than a standard elective
  - Two notches more generous than a standard elective
- Is that a good fit for your schedule?
- “Add this course!”

# Course Structure

- Course has **theoretical** and **practical** aspects
  - Best of both worlds!
- Need both in programming languages!
- **Reading = both**
  - Many external and optional readings
- **Review Sets = theory**
  - Not graded, practice problems for exams
- **Programming assignments = practice**
  - Electronic hand-in
- **Strict deadlines**
  - Ask me why ...

# Resources

- Textbook
  - Programming Language Pragmatics
  - Michael L. Scott, third edition
- Video Guides
- Free Online Materials
  - Udacity CS 262
- Optional Readings



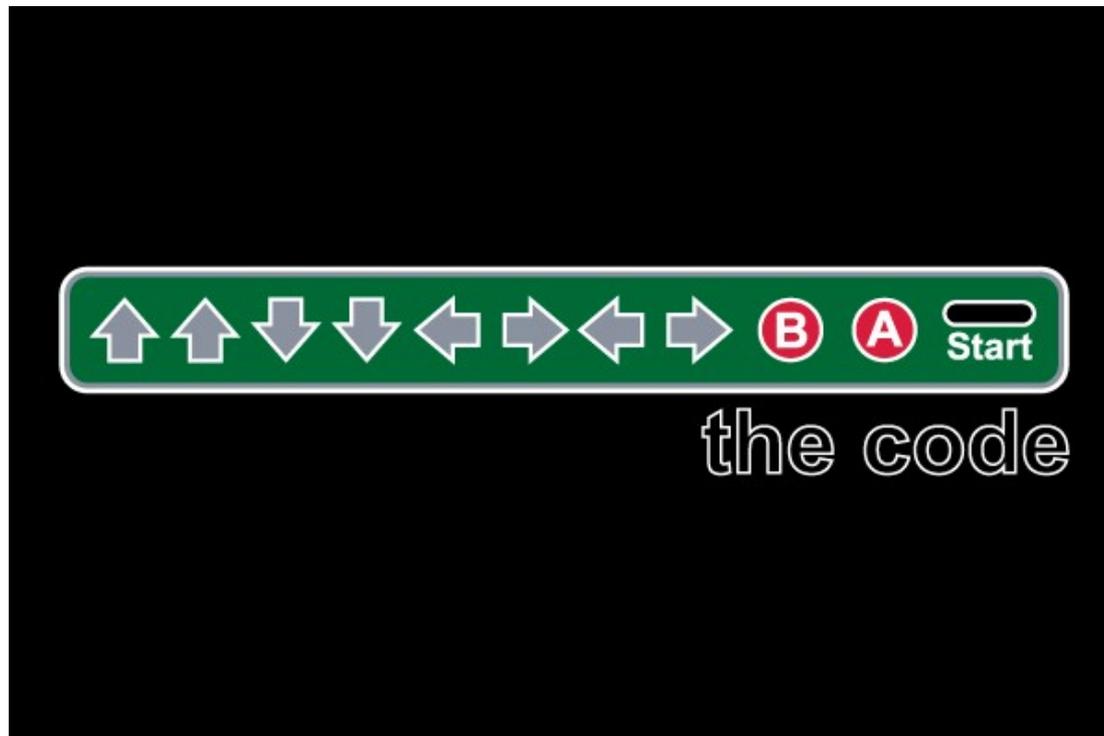
# Academic Honesty

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



# Academic Honesty

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

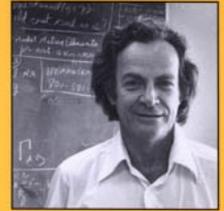


# LDI Course Project

- A big project: an Interpreter!
- ... in four easy parts
- You may optionally work in pairs.

## SIX EASY PIECES

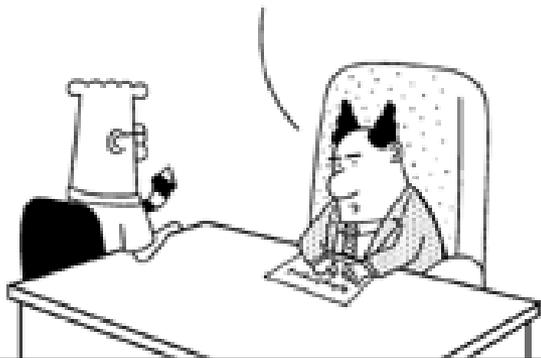
*Essentials of  
Physics  
Explained by  
Its Most  
Brilliant  
Teacher*



**RICHARD P.  
FEYNMAN**

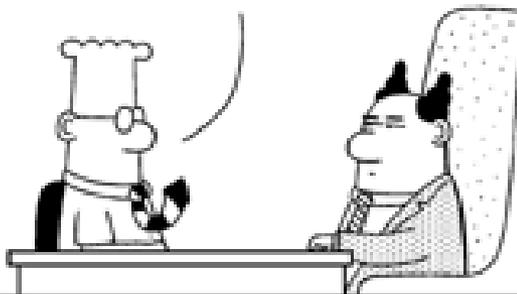
*Introduction by Paul Davies  
Author of The Mind of God*

WHAT DOES MFU2  
MEAN ON YOUR  
TIMELINE?



www.dilbert.com  
scottadams@aol.com

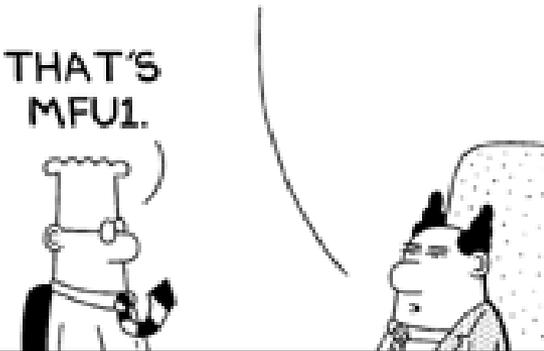
THAT'S MANAGEMENT  
FOUL-UP NUMBER TWO.  
IT USUALLY HAPPENS  
AROUND THE THIRD  
WEEK.



© 2006 Scott Adams, Inc./Dist. by UFS, Inc.

WE DON'T ANTICIPATE  
ANY MANAGEMENT  
MISTAKES.

THAT'S  
MFU1.

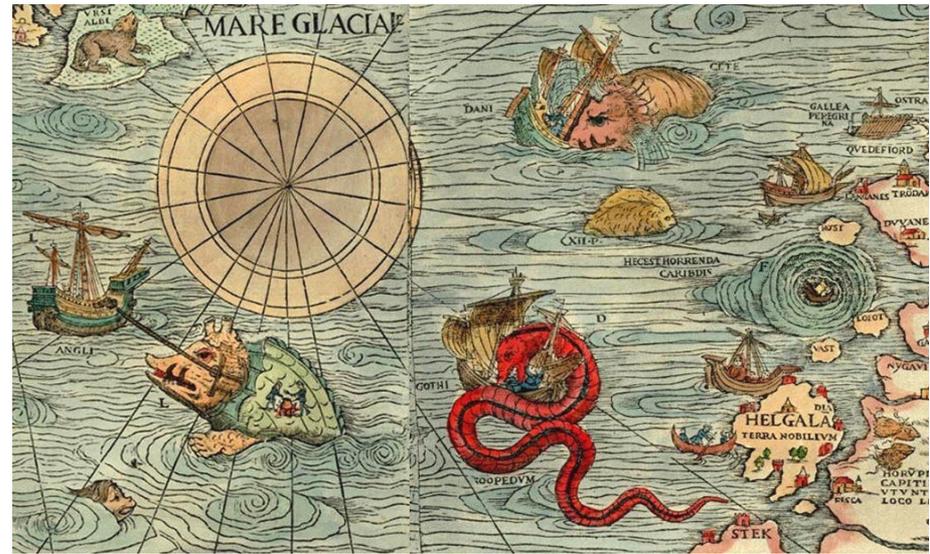


# Compilers Practicum Section

- There will be one sixty-minute Compilers Section each week (about 10 sections)
  - Hosted by Kevin Leach
- When do those meet?
  - Vote for times that everyone can make
  - Notes posted on web.
- Course will only fire if 12+ stay signed up

# “Explaining Unicorns”

- Visual Studio, JVM, Exceptions, Memory, Debugging, Linking, Shared Libraries, ...



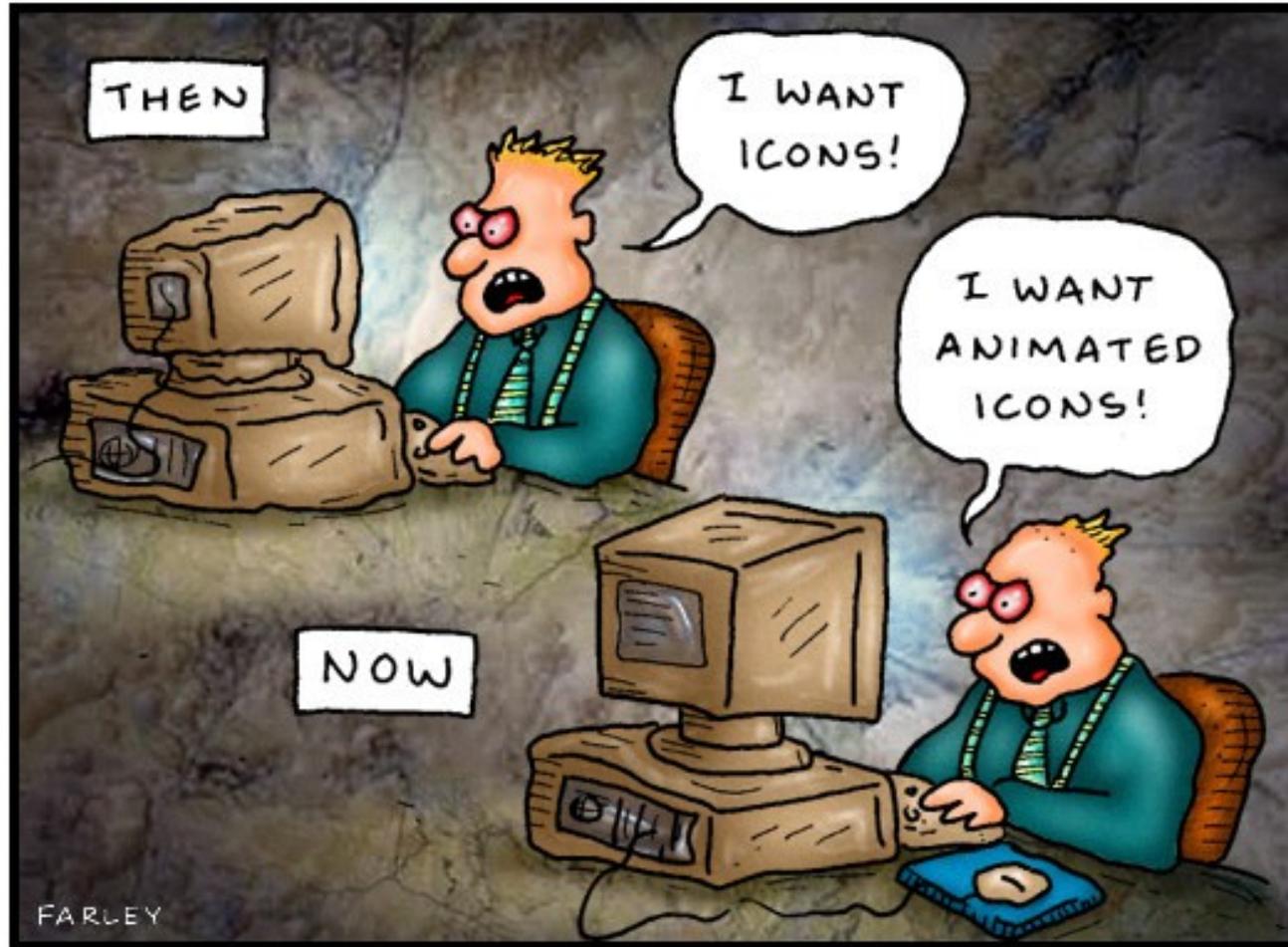
# How are Languages Implemented?

- 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

# Don't We Already Have Compilers?

## DOCTOR FUN

19 Mar 97



Copyright © 1997 David Farley, d-farley@tezcat.com  
<http://sunsite.unc.edu/Dave/drfun.html>

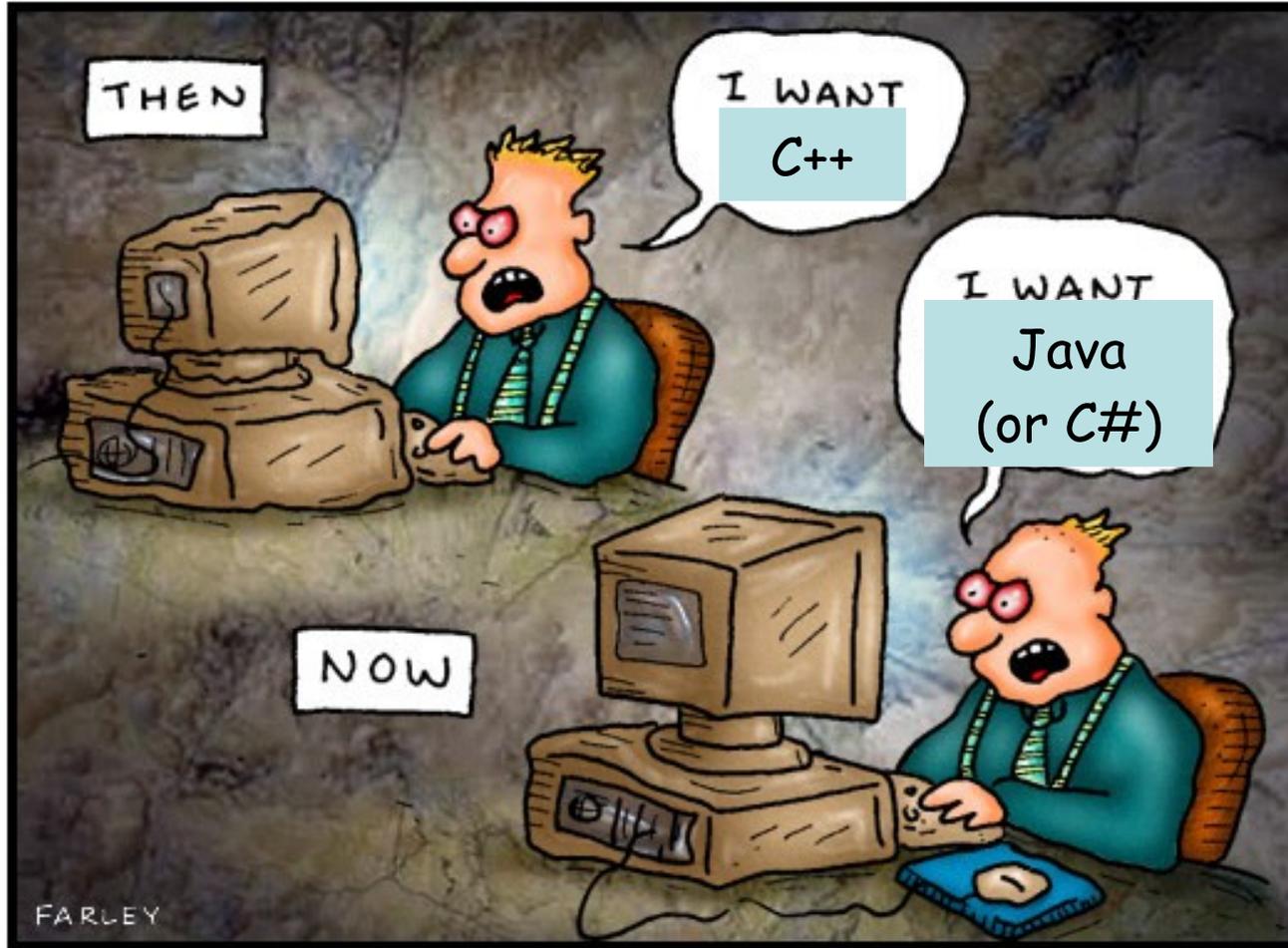
This cartoon is made available on the Internet for personal viewing only.  
Opinions expressed herein are solely those of the author.

Progress

# Dismal View Of Prog Languages

## DOCTOR FUN

19 Mar 97



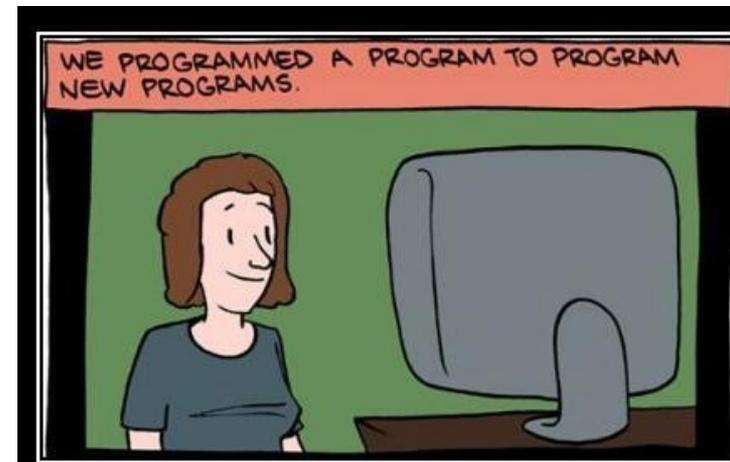
Copyright © 1997 David Farley, d-farley@tezcat.com  
<http://sunsite.unc.edu/Dave/drfun.html>

This cartoon is made available on the Internet for personal viewing only.  
Opinions expressed herein are solely those of the author.

Progress

# (Short) History of High-Level Languages

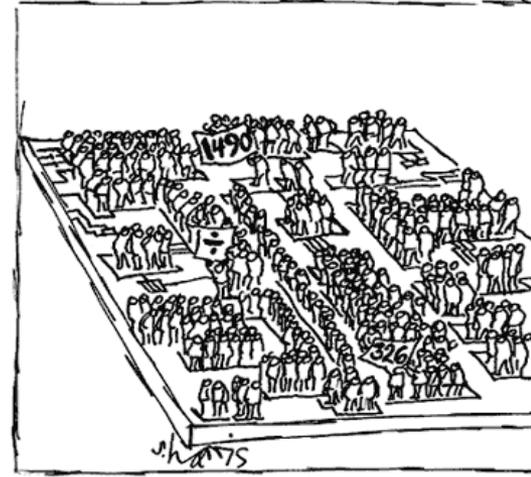
- 1953 IBM develops the 701 “Defense Calculator”
  - 1952, US formally ends occupation of Japan
  - 1954, Brown v. Board of Education of Topeka, Kansas
- All programming done in assembly
- Problem: Software costs exceeded hardware costs!
- John Backus: “Speedcoding”
  - An interpreter
  - Ran 10-20 times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

HUMAN SILICON CHIP:  
CAPABLE OF 6 COMPUTATIONS PER HOUR



# 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 keep the outlines of FORTRAN I



# Real-World Languages

- This Indo-European language is associated with South Asian Muslims and is the lingua franca of Pakistan. It developed from Persian, Arabic and Turkic influences over about 900 years. Poetry in this language is particularly famed, and is a reported favorite of US President Barack Obama.
- Example: السلام عليكم

# Interpreters

Lexical Analysis

Parsing

Semantic Analysis

Optimization (optional)

Interpret The Program

# Compilers

Lexical Analysis

Parsing

Semantic Analysis

Optimization (optional)

Generate Machine Code

Run that Machine Code

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

# Lexical Analysis

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

This is a sentence.

- Note the
  - Capital
  - Blank
  - Period

“T” (start of sentence symbol)

“ ” (word separator)

“.” (end of sentence symbol)



# 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



# And More Lexical Analysis

- Lexical analyzer divides program text into “words” or tokens

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

- Broken up:

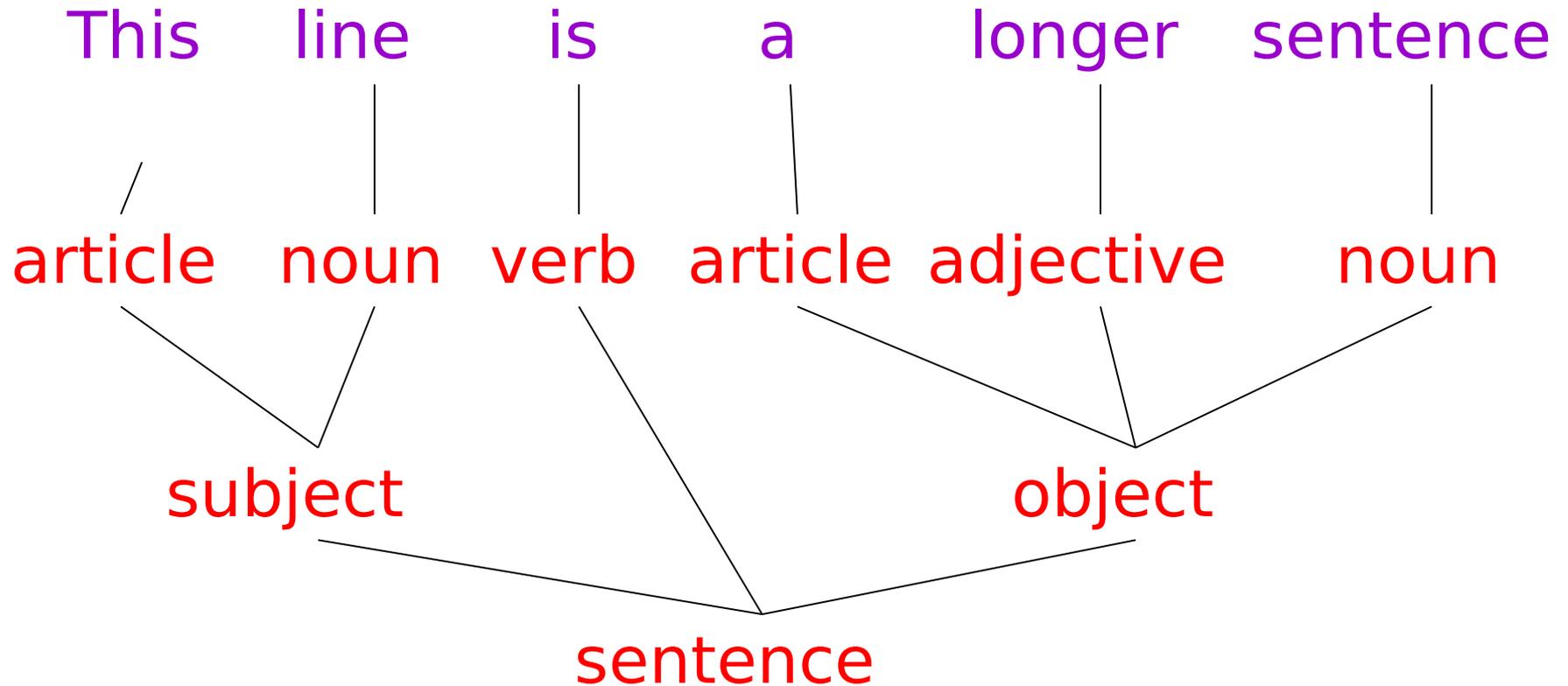
if, x, ==, y, then, z, =, 1, ;, else, z, =, 2, ;

# Parsing

- Once words are understood, the next step is to understand sentence structure
- Parsing = Diagramming Sentences
  - The diagram is a tree
  - Often annotated with additional information



# Diagramming a Sentence

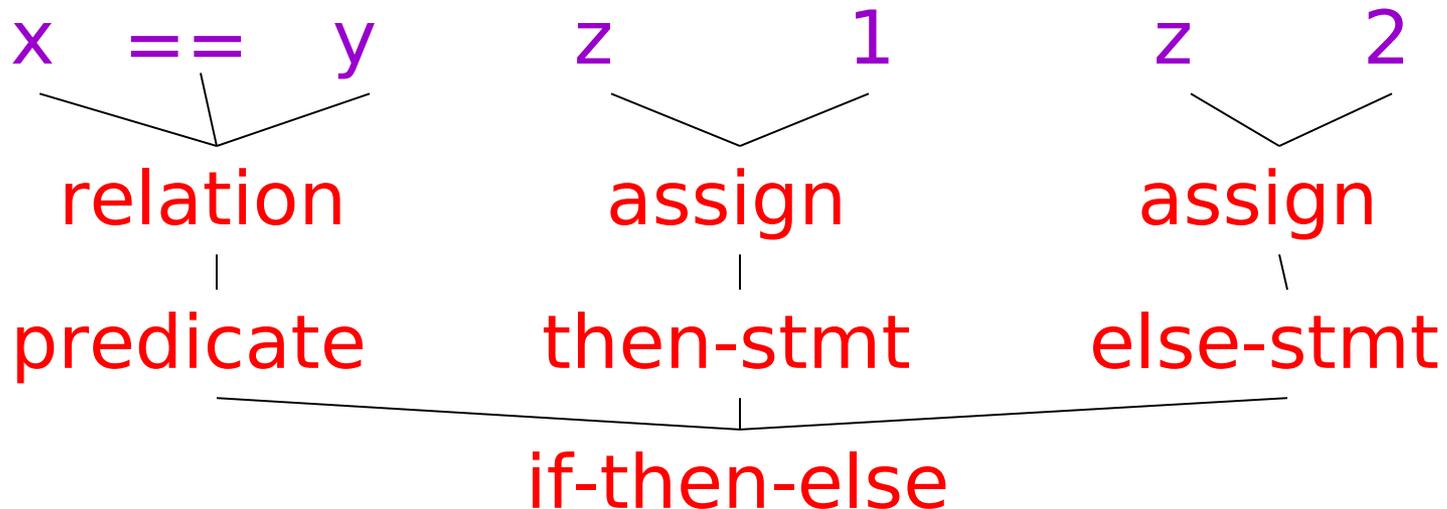


# Parsing Programs

- Parsing program expressions is the same
- Consider:

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

- Diagrammed:



# 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

# 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?

It's  
context-  
sensitive!

# Semantic Analysis in Programming

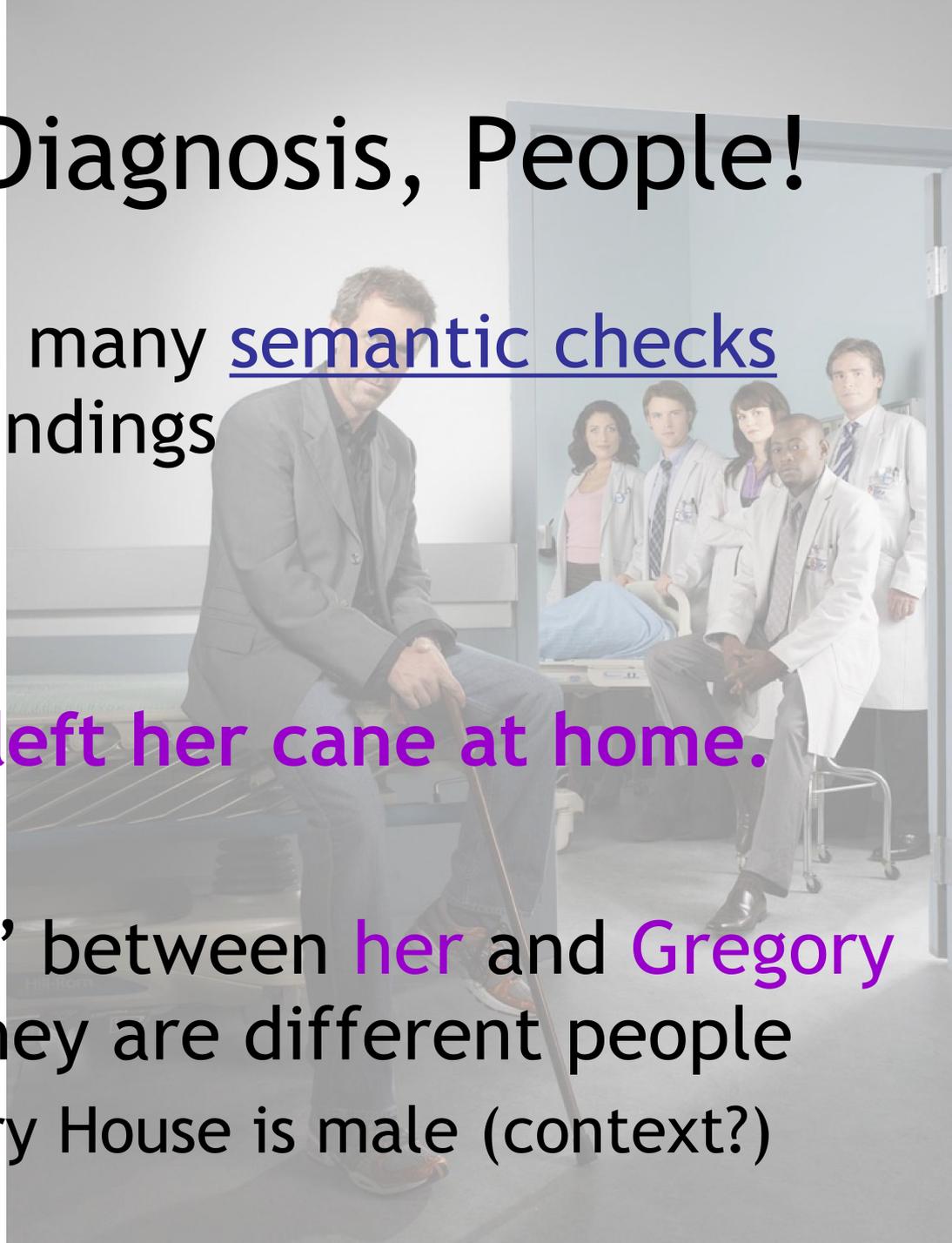
- Programming languages define strict rules to avoid such ambiguities
- This C++ code prints “4”; the inner definition is used

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



# Differential Diagnosis, People!

- 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 think they are different people
  - Presumably Gregory House is male (context?)



# Optimization

- No strong counterpart in English, but akin to editing (cf. poems, short stories)
- **Automatically modify programs** so that they
  - Run faster
  - Use less memory
  - In general, conserve some resource
- “Compilers” studies Optimizations in depth

# Code Generation

- Produces assembly code (usually)
  - which is then assembled into executables by an assembler
- A translation into another language
  - Analogous to human translation
- “Compilers”: produce machine code
  - Either “Java Bytecode” or x86-64 assembly

# Issues

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



# 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
  - Thus: this course puts no emphasis on ancient parsing optimizations (e.g., LL, LALR)

# 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
  - ...

# Why Study Prog. Languages?

- Prepare for many good jobs
- Increase capacity of expression
- 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
- Computers are the only tools that increase cognitive power, so learn to control them

# What Will You Do In This Class?

- **Reading** (textbook, videos, outside sources)
- **Learn** about different kinds of languages
  - Imperative vs. Functional vs. Object-Oriented
  - Static typing vs. Dynamic typing
  - etc.
- Gain exposure to new languages (ML, Cool)
- **Write an interpreter!**

# What Is This?

A lungo il mio cuore di tali ricordi ha voluto colmarsi!

Come un vaso in cui le rose sono state dissetate:

Puoi romperlo, puoi distruggere il vaso se lo vuoi,

Ma il profumo delle rose sarà sempre tutt'intorno.

Długo, długo moje serce przepelnione było takimi wspomnieniami!

Były jak waza, w której kiedyś róże destylowały:

Możesz sprawić by pękła, możesz gruchotać wazę jeśli chcesz,

Ale zapach róż będzie wciąż czuć dookoła.

Mon coeur est brûlant rempli de tels souvenirs

Comme un vase dans lequel des roses ont été distillées:

Tu peux le briser, tu peux détruire le vase si tu le désires,

Mais la senteur des roses sera toujours là.

Lang, lang soll die Erinnerung in meinem Herzen klingen!

Gleich einer Vase, drin Rosen sich einst tränkten:

Lass sie zerbrechen, lass sie zerspringen,

Der Duft der Rose bleibt immer hängen.

Muito, muito tempo seja meu coração preencho com tais lembranças!

Tal qual o vaso onde rosas foram uma vez destiladas:

Pode quebrar, pode estilhaçar o vaso se o desejas,

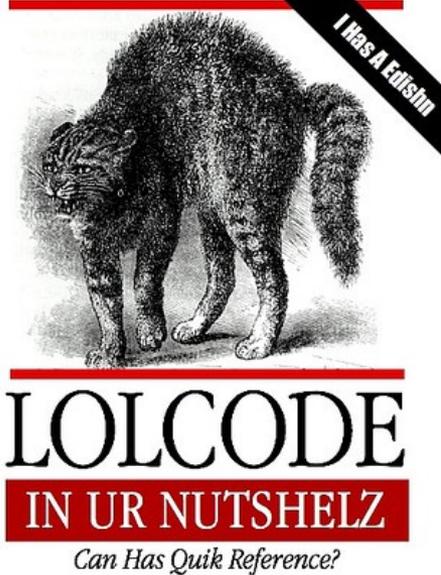
Mas perdurará para sempre o aroma das rosas perfumadas.

# The Rosetta Stone

- The first programming assignment involves **writing the same simple (50-75 line) program in two languages:**
  - **Ocaml and Cool (with Ruby, Python, JavaScript, Haskell and C as Extra Credit)**
- PA1c, **due Thu Jan 28**, requires you to write the program in one language
- PA1, due subsequent Tue, requires both

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)



O'RLY

Charlie Mab Bucket

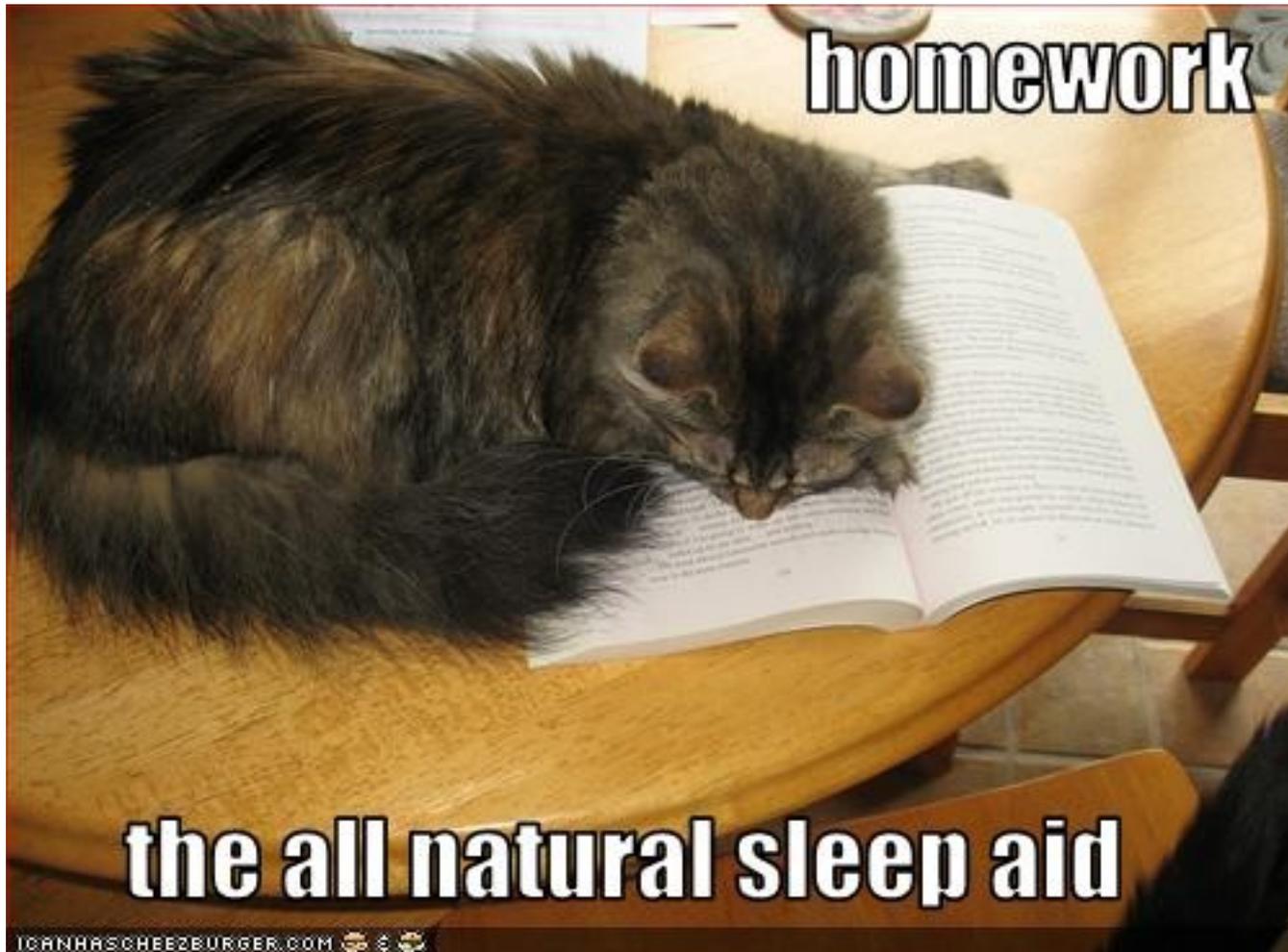
# Live Submission Demo

- Let's visit the automated submission website



# Start The Homework Now

- We can help you!



# 2-Credit “Sidecar” Course: Compilers Practicum

- The final project for the Compilers Practicum is a working **optimizing compiler** (code generator) that produces either bytecode or x86-64 assembly.
- At the end of this course, you will be acquainted with the fundamental concepts in code generation and optimization. In particular, you will understand the **theory and practice of code generation, stack layouts, calling conventions, dynamic dispatch, control flow graphs, and dataflow analyses.**

# Compilers Practicum Section

- There will be one sixty-minute Compilers Section each week (about 10 sections)
  - Hosted by Kevin Leach
- When do those meet?
  - *Vote today* for times that everyone can make
  - Notes posted on web
- Course will fire if 12+ stay signed up

# Homework

- Scott Book reading (for Tuesday)
- Get started on PA1c (due in 7 days)

## Questions?

