Welcome
- You are in comp181 – Compilers
- Lectures: Tuesday, Thursday 1:30 to 2:45
  - If you’re here, you probably know this
  - Will be on the web page... but, I urge you to attend class
- Class web page: [www.cs.tufts.edu/~sguyer/classes/comp181](http://www.cs.tufts.edu/~sguyer/classes/comp181)
- Mailing list: [www.eecs.tufts.edu/mailman/listinfo/comp181](http://www.eecs.tufts.edu/mailman/listinfo/comp181)
- Book: None right now, but stay tuned...

Introduction
- What is this artifact?  
  - The Rosetta Stone
- Significance?  
  - Same document in Greek and Egyptian hieroglyphics
- Why are you telling us this?  
  - Compilers are language translators

A Brief History of High-Level Languages
- 1953 IBM develops the 701
  - Memory: 4096 words of 36 bits
  - Speed: 60 msec for addition
  - All programming done in assembly code
- Problem: Software costs exceeded hardware costs!
- John Backus: “Speedcoding”  
  - Simulate a more convenient machine
  - But, 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  
    - Had already failed in other projects
- 1954-7 FORTRAN I project  
  - By 1958, >50% of all code is in FORTRAN
  - Cut development time dramatically  
    - From weeks to hours
Languages involved

Variables vs registers

Control vs jumps

Source

Target

Languages involved

Languages involved

The compilation problem

- Assembly language
  - Converts trivially into machine code
  - No abstraction: load, store, add, jump, etc.
  - Extremely painful to program

- High-level language
  - Easy to understand and maintain
  - Abstractions: control (loops, branches); data (variables, records, arrays); procedures

Problem: how to implement the language?

Language implementations

- Two major strategies:
  - Interpretation
  - Compilation

- Interpreter
  - "Online": read program, execute immediately
  - Examples?

- Compiler
  - "Offline": convert high-level program into assembly code
  - Examples?

- Compilation is a language translation problem

How does translation work?

Can you think of another strategy—a "hybrid"?

Sounds easy!

- Translation can be tricky...
  - Infallible source: the Internet

  I saw the Pope ("el Papa")
  I saw the potato ("la papa")

  It won't leak in your pocket and embarrass you ("no los embarazos")
  It won't leak in your pocket and make you pregnant ("no embarazado")

  It takes a tough man to make a tender chicken
  It takes a hard man to make a chicken affectionate

Job #1

- What is our primary concern?
  - Words or code: translate it correctly

- How do we know the translation is correct?
  - Specifically, how do we know the resulting machine code does the same thing

- "Does the same thing"
  - What does that even mean?
Correctness
- **Practical solution**: automatic tools
  - Parser generators, regular expressions, rewrite systems, dataflow analysis frameworks, code generator-generators
  - Extensive testing
- **Theoretical solution**: a bunch of math
  - Formal description of semantics
  - A proof that the translation is correct
  - Topic of current research

Incorrectness
- **What is this?** The infamous “Blue Screen of Death”
- **Internal failure in the operating system**
- **Buggy device driver**

Good enough?
- **Is there more than correctness?**

  **Our wines leave you nothing to hope for.**
  - Swiss menu

  **When passenger of foot heave in sight, tootle the horn. Trumpet him melodiously at first, but if he still obstacles your passage then tootle him with vigor.**
  - Car rental brochure

  **Drop your pants here for best results.**
  - Tokyo dry cleaner

Job #2
- **Produce a “good” translation**
- **What does that mean for compilers?**
  - Good performance – optimization
  - Find more efficient code sequences
  - Utilize the hardware effectively
- **How hard could that be?**

Modern processors
...are very, very complex...
- Pentium 4
- Xbox 360
- PS-3 CELL

Study of compilers
- **Brings together many parts of CS**
  - Practical and theoretical
  - Some solved problems, others unsolved
Structure of a compiler

- Organized as a series of passes
  - Lexical Analysis
  - Parsing
  - Semantic Analysis
  - Optimization
  - Code Generation

- We will follow this outline in the class

Compilers Today

- The overall structure of almost every compiler adheres to our outline
- Emphasis has changed since FORTRAN:
  - Early
    - Lexing, parsing most complex, expensive
  - Today
    - Lexing and parsing are cheap
    - Good techniques for code generation
    - Optimization dominates all other phases

Related systems

- Interpreters
- Scripting languages: Javascript, Ruby, etc.
- XML processing
- Just-in-time compilers
  - Java JVM
  - Microsoft CLR
- Binary instrumentation
  - PIN
  - valgrind

Trends in Compilation

- Compilation for speed is less interesting. But:
  - Scientific computing, parallel computing
  - Advanced processors
  - Digital signal processors, multi-core
  - Implementation of modern languages (Java, C#)
    - Good performance is a challenge

- Compilation for improving code reliability:
  - Memory safety
  - Detecting concurrency errors (data races)
  - Detecting security vulnerabilities

Course Structure

Course has theoretical and practical aspects

- Programming assignments = practice
  - Five part project – more about this next time
  - 50% of final grade
- Written assignments = theory, practice
  - Class hand-in, right before lecture
  - 15% of final grade
- Midterm exam: 15%
- Final exam: 20%

Late policy:
Three free late days for assignments, use them however you want
Next time...

- Sign up on mailing list:
  https://www.eecs.tufts.edu/mailman/listinfo/comp181
- The view from 35,000 feet
- Information about projects