Masks are optional, but recommended–Covid is still with us.
Your colleagues (including course staff) thank you.

CSE 401/M501 – Compilers

Overview and Administrivia

Larry Ruzzo

Spring 2022

CSE 401/M501

This is the home page for CSE 401 / CSE M 501. This, and the navigation bar at the top of this page, present important course content and resources.

News

3/27: Welcome to CSE 401 and CSE M 501, Spring Quarter, 2022.

3/27: This website, excluding the Message Board link, is "student-ready", although additional items will trickle in. Please email `]cs` if you see ambiguities, contradictions, missing or broken links, etc.

3/27: Masks, Lecture Streaming, etc.: especially note the info on lecture access in the "Lectures" paragraph bel ed there.

Staff

Instructor: Larry Ruzzo.

Teaching Assistants: Robert Burris, Morel Takougang, Jack Zhang, and Apollo Zhu.

Contact Info:

- Use the message board (linked from the navigation bar at the ost questions about course content or assignments. If you have a question, so do 20 of your classmates; this makes the discussion visible to
- *Feel free to answer other students' questions*; stude isper than my own.
- "Private" messages on the discussion board a , appropriate for questions containing, e.g., putative answers or detailed code.
- For other issues, (health, personal circu ther concerns) email `cse401-staff[at]cs` (this reaches Ruzzo and all TAs; despite the name, for use it for CSE M 501, too), or if you pref `[at]cs`. This will help us track things to best help you.

Activities

Lectures: MWF 2:30 tures will be in-person and "livestreamed" via Panopto ; see Masks, Livestreams, and Recordings for more on this. I will try to monitor Pano remote viewer's questions during lecture (but, honestly, I don't always see them...). Recordings will also be available on-demand via Panopto ther items will be posted, generally before each class, linked from the Calendar page.

fternoons. Sections will not be recorded, but we will make occasional videos of tutorial material. All section materials will be linked from the Calendar

Hours: TBD. Likely a mix of in-person and Zoom.

All in-person activities will follow then-current UW guidelines regarding masks, etc. Please be respectful of others' neeeds.

Agenda

- Introductions
- Administrivia
- What's a compiler?
- Why you want to take this course ☺

Logistics

- It's (mostly) in-person this quarter
  - Masks (mostly) optional but still recommended – be smart, stay healthy, respect others' needs, avoid crowding in office hours, etc.
  - Lectures (but not sections) will be recorded and live-streamed on Panopto; suit yourself.
  - In-person midterm + final planned
  - In-person + Zoom office hours

Stay in Touch – Speak Up!

- This is a strange world we're (still) in and there's (still) a lot of stress for many people (although maybe different now)

- Please speak up if things are/aren't going well
  - We can't help if we don't know; stay in touch with TAs, instructor, advising, friends, peers, family

- We're all in this together but not all in the same way, so please show understanding and compassion for each other and help when you can – both in and outside of class

Who: Course staff

- Instructor: Larry Ruzzo: UW faculty for a while; CSE 401 7x veteran (but not for a long time)
- TAs: Robert Burris, Morel Takougang, Jack Zhang, Apollo Zhu
- Get to know us – we're here to help you succeed!
- Office Hrs: Probably a combination of Zoom and in-person. Starting ASAP – watch for announcements.

Credits

The course builds on many years of successful offerings by many faculty here and at other schools, most importantly by UW's Prof. Hal Perkins, but I won't attempt to provide detailed attributions.

CSE M 501

- Enhanced version for 5th-year BS/MS students.
- 401 and M501 share a lot: the same lectures, sections, assignments, infrastructure, …
- The key difference is that M501 students will have to do a significant addition to the project; see course website for details.

Background

- Official prerequisites:
  - CSE 332 (data abstractions)
    - and therefore CSE 311 (Foundations)
  - CSE 351 (hardware/software interface, x86_64)
- Also very useful, but not required:
  - CSE 331 (software design & implementation)
  - CSE 341 (programming languages)
  - Who's taken these?

Lectures & Sections

- Both valuable
- All material posted, but be there! Take notes!
- Lectures: in-person, live-streamed, recorded
- Sections: NOT recorded. Additional examples and exercises plus project details and tools
  - We will have sections this week (Thursday).
    - Watch time schedule/email/Ed for possible room changes!

Communications

- Course web site (www.cs.uw.edu/401)
- Discussion board – Ed (coming asap)
  - For anything related to the course
  - Join in! Help each other out. Staff will contribute.
  - Also use for private messages with too-specific-to-post questions, code, etc.
  - Staff will also use to post announcements
- Email to cse401-staff[at]cs for things not appropriate for Ed, that need followup, …

Grading

- I plan to have normal in-person midterm and final exams
- Roughly:
  - 50% project, done with a partner
  - 20% individual written homework
  - 10% midterm
  - 20% final

  May be adjusted as needed/appropriate

Academic Integrity

- We want a collegial group helping each other succeed!
- But: you must never misrepresent work done by someone else as your own, without proper credit if appropriate, or assist others to do the same
- Read the course policy carefully
- We trust you to behave ethically
  - I have little sympathy for violations of that trust
  - Honest work is the most important feature of a university (or engineering or business or life). Anything less disrespects your instructor, your colleagues, and yourself

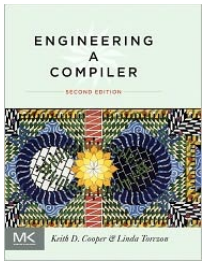
Course Project

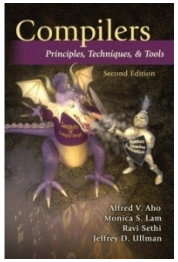
- Best way to learn about compilers is to build one!
- Course project: MiniJava compiler
  - Core parts of Java – classes, objects, etc.
  - Generate executable x86-64 code & run it
  - Completed in steps through the quarter
    - End point is the most important part, but intermediate milestones keep you on schedule and allow feedback
  - Additional work for CSE M 501 students – see web

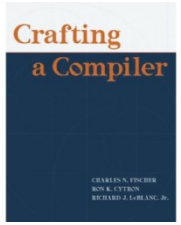
Project Groups

- You should work in pairs
  - Pick a partner now to work with throughout quarter – we need this info by early next week
  - CSE M 501 students should pair up with someone else in M501 (401 → M 501 switches are possible if it makes sense for individual(s) involved)
  - Collaboration via internet works surprisingly well, so you don't need to (but may) hang out together in the labs.
- We'll provide per-group accounts on department gitlab server for groups to store and synchronize their work; we'll pull files from there for project feedback / grading
  - Anybody new to CSE Gitlab/Git?

Books





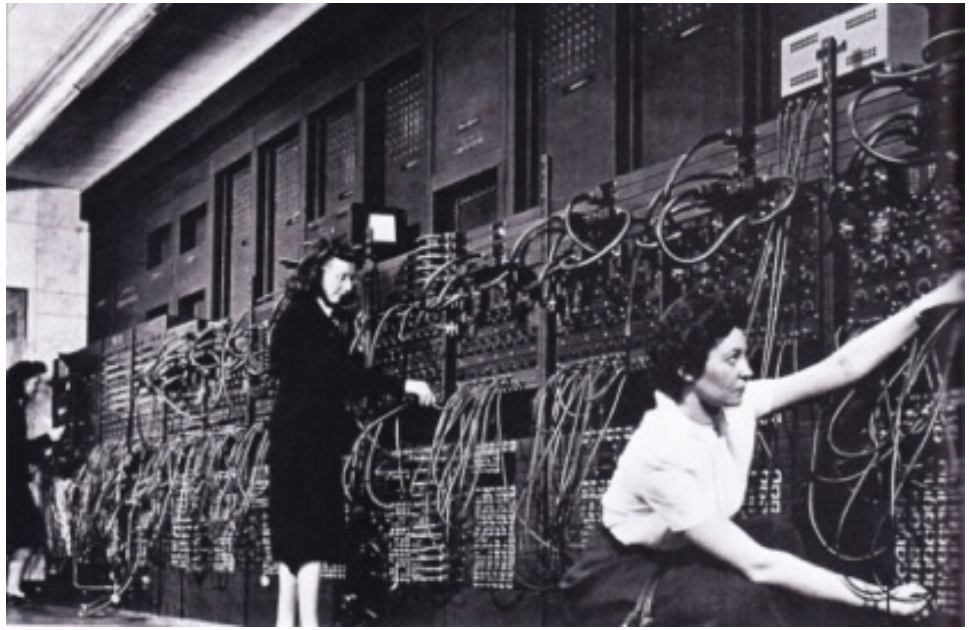


- Official text:
  - Cooper & Torczon, *Engineering a Compiler.* *Available free online through UW Library Safari books subscription.* See syllabus.
- Three other good books; enrichment & alternative perspectives
  - Appel, *Modern Compiler Implementation in Java*, 2nd ed. MiniJava is from here.
  - Aho, Lam, Sethi, Ullman, "Dragon Book"
  - Fischer, Cytron, LeBlanc, *Crafting a Compiler*

Compilers: What/Why

- "Algorithms": communications between people about computations
- "Programs": communications between people and machines about computations

```
int nPos = 0;
int k = 0;
while (k < length) {
   if (a[k] > 0) {nPos++;}
}
```

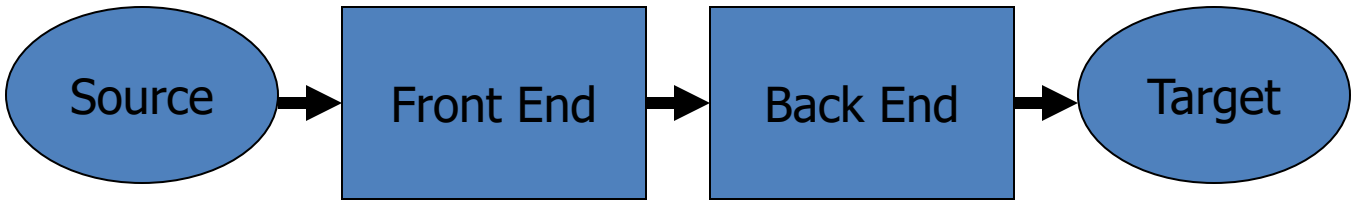


Programming The Eniac

- Compilers automate tedious and error-prone detail-work to bridge the people-facing end of that conversation to the deeply rigid 0-1 world at the other end. (Where's k? Where's a[k]? Is it >0? …)

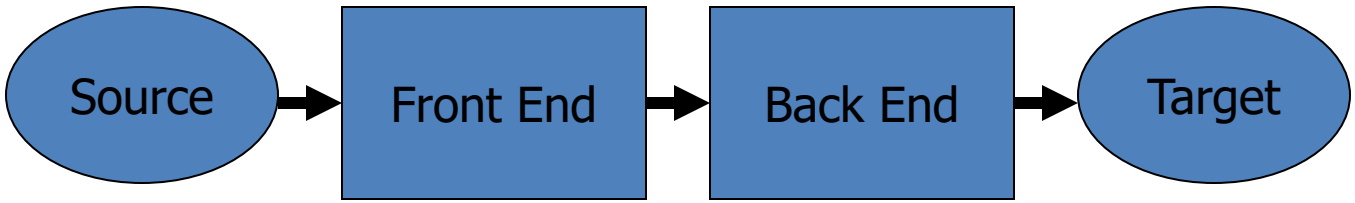
Structure of a Compiler

- At a high level, a compiler has two pieces:
  - Front end: analysis
    - Read source program and discover its structure and meaning
  - Back end: synthesis
    - Generate equivalent target language program



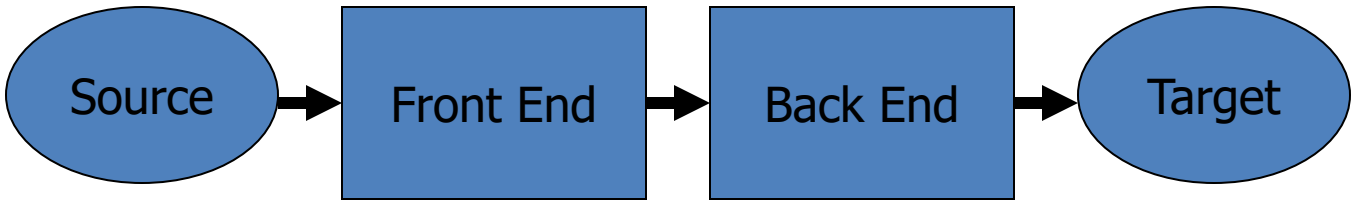
Compiler must…

- Recognize legal programs (& complain about illegal ones)
- Generate correct code
  - Compiler can attempt to improve ("optimize") code, but must not change behavior (meaning)
- Manage runtime storage of all variables/data
- Agree with OS & linker on target format

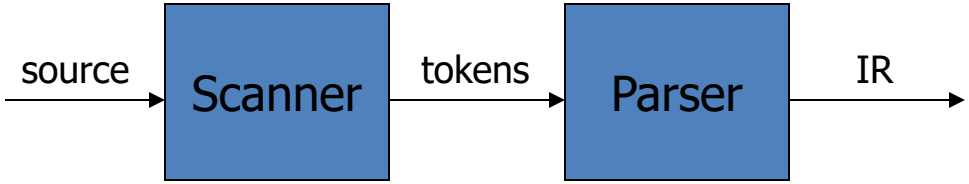


Implications

- Phases communicate using some sort of Intermediate Representation(s) (IR)
  - Front end maps source into IR
  - Back end maps IR to target machine code
  - Often multiple IRs – higher level at first, lower level in later phases



Front End



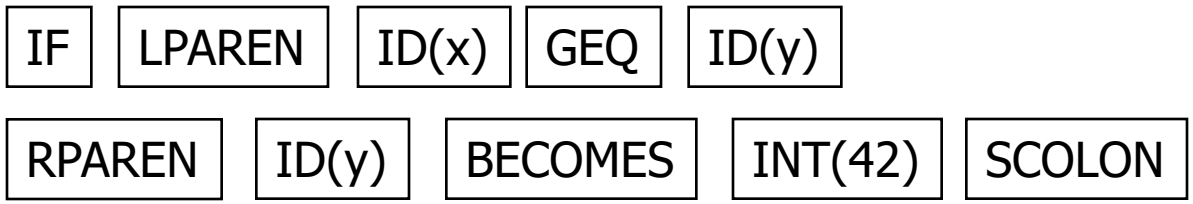
- Usually split into two parts
  - Scanner: Responsible for converting character stream to token stream: keywords, operators, variables, constants, …
    - Also: strips out white space, comments
  - Parser: Read token stream; validate structure; generate IR
    - Either here or shortly after, perform semantics analysis to check for things like type errors, etc.
- Both of these can be generated automatically
  - Use a formal grammar to specify the source language
  - Tools read the grammar and generate scanner & parser (lex/yacc or flex/bison for C/C++, JFlex/CUP for Java)

Scanner Example

- Input text

```
// this statement does very little
if (x >= y) y = 42;
```

- Token Stream



  - Notes: tokens are atomic items, not character strings; comments & whitespace are *not* tokens (in most languages – counterexamples: Python indenting, Ruby and JavaScript newlines)
    - Token objects sometimes carry associated data (e.g., numeric value, variable name)

Parser Output (IR)

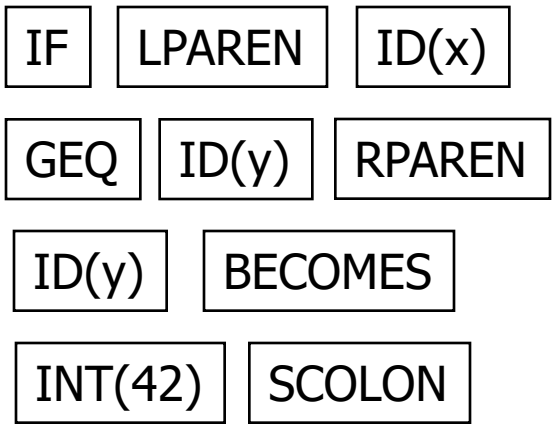
- Given token stream from scanner, the parser must produce output that captures the meaning of the program
- Most common parser output is an abstract syntax tree (AST)
  - Essential meaning of program without syntactic noise
  - E.g., nodes are operations, children are operands
- Many different forms
  - Engineering tradeoffs change over time
  - Tradeoffs (and IRs) can also vary between different phases of a single compiler

Scanner/Parser Example

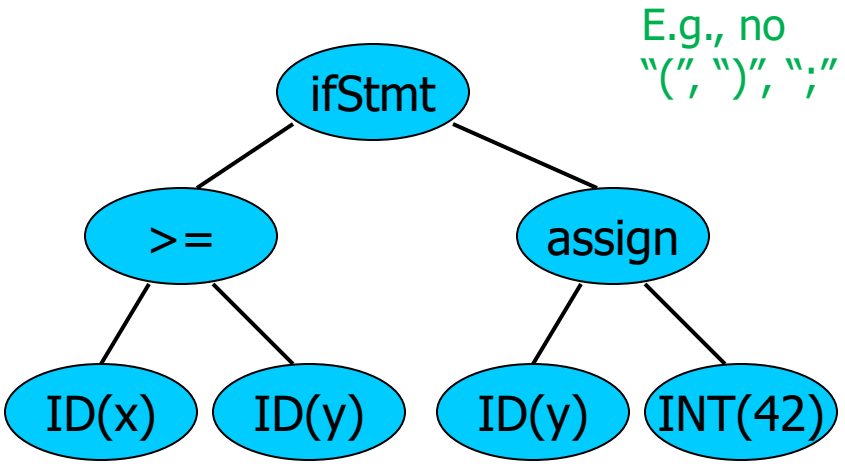
Original source program:

```
// this statement does very little
if (x >= y) y = 42;
```

- Token Stream



- Abstract Syntax Tree



Static Semantic Analysis

- During or (usually) after parsing, check that the program is legal and collect info for the back end
  - Type checking
  - Verify language requirements like proper declarations, etc.
  - Preliminary resource allocation
  - Collect other information needed by back end analysis and code generation
- Key data structure: Symbol Table(s)
  - Maps names -> meaning/types/details

Back End

- Responsibilities
  - Translate IR into target code
  - Should produce "good" code
    - "good" = fast, compact, low power (pick some)
  - Should use machine resources effectively
    - Registers
    - Instructions
    - Memory hierarchy

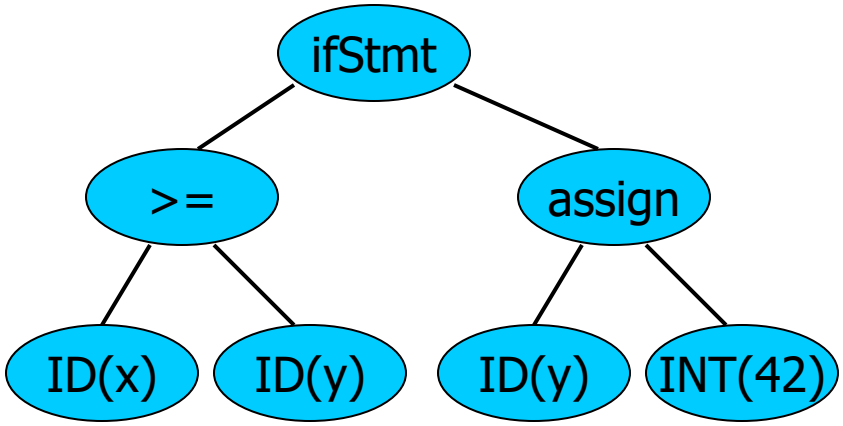
Back End Structure

- Typically two major parts
  - "Optimization" – code improvement – change correct code into semantically equivalent "better" code
    - Examples: common subexpression elimination, constant folding, code motion (move invariant computations outside of loops), function inlining (replace call with body of function)
    - Optimization phases often interleaved with analysis
  - Target Code Generation (machine specific)
    - Instruction selection & scheduling, register allocation
- Usually walk the AST and generate lower-level intermediate code before optimization

The Result

- Input

```
if (x >= y)
    y = 42;
```



- Output

```
    movl  16(%rbp),%edx
    movl  -8(%rbp),%eax
    cmpl   %eax, %edx
    jl       L17
    movl  $42, -8(%rbp)
L17:
```

Why Study Compilers? (1)

- Become a better programmer(!)
  - Insight into interaction between languages, compilers, and hardware
  - Understanding of implementation techniques, how code maps to hardware; fast/slow/mem hungry?
  - Better intuition about what your code does
  - Understanding how compilers optimize code helps you write code that is easier to optimize
    - And avoid wasting time doing "optimizations" that the compiler will do better, and avoid "clever" code that confuses the compiler and makes thing worse

Why Study Compilers? (2)

- Compiler techniques are everywhere
  - Parsing ("little" languages, program input, scripts,…)
  - Software tools (verifiers, checkers, …)
  - Database engines, query languages
  - Domain-specific languages, ML, data science
  - Text processing
    - Tex/LaTex -> dvi -> Postscript -> pdf
  - Hardware: VHDL; model-checking tools
  - Mathematics (Mathematica, Matlab, SAGE)

Why Study Compilers? (3)

- Fascinating blend of theory and engineering
  - Lots of beautiful theory around compilers
    - Parsing, scanning, static analysis
  - Interesting engineering challenges and tradeoffs, particularly in optimization (code improvement)
    - Ordering of optimization phases
    - What works for some programs can be bad for others
  - Plus some very difficult problems (NP-hard or worse)
    - E.g., register allocation is equivalent to graph coloring
    - Need to come up with "good enough" approximations / heuristics

Why Study Compilers? (4)

- Draws ideas from many parts of CSE
  - AI: Greedy algorithms, heuristic search
  - Algorithms: graphs, dynamic programming, approximation
  - Theory: Grammars, DFAs and PDAs, pattern matching, fixed-point algorithms
  - Systems: Allocation & naming, synchronization, locality
  - Architecture: pipelines, instruction set use, memory hierarchy management, locality

Why Study Compilers? (5)

- You might even write a compiler some day!
- You ***will*** write parsers and interpreters for little languages, if not bigger things
  - Command languages, configuration files, XML, JSON, network protocols, …
- Don't be surprised if it's handy in a job interview
- And if you like working with compilers and are good at it there are many jobs available…

Any questions?

- Your job is to ask questions to be sure you understand what's happening and to slow me down
  - Otherwise, I'll barrel on ahead ☺

Coming Attractions

- Quick review of formal grammars
- Lexical analysis – scanning & regular expressions
  - Background for first part of the project
  - Next 2-3 lectures + Thursday's sections
- Followed by parsing … Start reading: ch. 1, 2.1-2.4
  - Entire book available through Safari Online to UW community – see syllabus for link

Before next time…

- Familiarize yourself with the course web site
- Esp. syllabus, academic integrity, calendar,
- Find a partner!
  - And meet other people in the class too!! ☺