Compilers

Advanced Systems Elective
Discussion question

- What is a compiler?

```
Error: obscure syntax mistake [main.cpp:375] !!!
```

"An angry translator."
-- previous CS 432 student
A compiler is a computer program that automatically translates other programs from one language to another

- (usually from a human-readable language to a machine-executable language, but not necessarily)
Automated translation

- Compilation vs. interpretation:

  Program in Source Language ➔ Compiler ➔ Program in Target Language

  Program in Source Language ➔ Interpreter ➔ Results
### "X is a compiler" alignment chart

<table>
<thead>
<tr>
<th>Input Purist</th>
<th>Output Purist</th>
<th>Output Neutral</th>
<th>Output Rebel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input must be a program</td>
<td>gcc is a compiler</td>
<td>prettier is a compiler</td>
<td>An orchestra is a compiler</td>
</tr>
<tr>
<td>Input Neutral</td>
<td>Microsoft Word is a compiler</td>
<td>Javadoc is a compiler</td>
<td>Al Dungeon is a compiler</td>
</tr>
<tr>
<td>Input Rebel</td>
<td>A coin flip is a compiler</td>
<td>Bop It! is a compiler</td>
<td>The sun is a compiler</td>
</tr>
</tbody>
</table>
Rhetorical question

• Why should we study compilers?
  - *(besides getting systems elective credit...)*
Compilers: a convergent topic

- Data structures
  - CS 240
- Architectures, machine languages, and operating systems
  - CS 261, CS 450
- Automata and language theory
  - CS 327, CS 430
- Graph algorithms
  - CS 327
- Software and systems engineering
  - CS 345, CS 361
- Greedy, heuristic, and fixed-point algorithms
  - CS 452
Reasons to study compilers

• Shows how many areas of CS can be combined to solve a truly "hard" problem (automated language translation)

• Bridges theory vs. implementation gap
  – Theory informs system development
  – We will never lose sight of our primary objective

• Practical experience with large(er) software systems
  – My master copy is over 4K LOC
  – Of this, you will re-write over 1K LOC this semester
  – Need to address software engineering concerns
Course goal

• Fundamental question
  - "How do compilers translate from a human-readable language to a machine-executable language?"

• After this course, your response should be:
  - "It's really cool! Let me tell you..."
Course design

• First, a bit of course design theory ...
Course design theory

- Big ideas
  - E.g., "A compiler is a large software system consisting of a sequence of phases"
- Themes (stuff you should remember in five years)
  - E.g, "Large problems can sometimes be solved by composing existing solutions to smaller problems."
- Learning objectives (stuff you should remember at the end of the course)
  - E.g., "Identify the technical challenges of building a large software system such as a compiler."

- Activities and assignments flow from learning objectives
  - E.g., "Draw a diagram illustrating the major phases of a modern compiler."
- Exams reflect activities and assignments
- Goal: “engaged” and effective learning
Evolution of CS 432

• Fall 2015 - special topics (CS 480)
  – Adaptation of CS 630 (graduate course) taught in Spring 2015

• Fall 2016 - first time taught as CS 432
  – First time teaching CS 261 as well

• Fall 2017
  – Expanded test suite significantly, added type systems and lambda calculus

• Fall 2018
  – Added Y86 translator to “close the loop” with CS 261, removed lambda calculus

• Fall 2019 (two sections)
  – Removed reflection paper assignments, switched to Dragon book for LR parsing

• Fall 2020
  – Re-wrote entire project in C (w/ re-worked grading scheme), transitioned to take-home exams

• Fall 2021
  – Added hybrid virtual/in-person office hours

• Fall 2022
  – Official support for VS Code + Remote SSH development platform
Course objectives

- Identify and discuss the technical and social challenges of building a large software system such as a compiler.
- Develop and analyze formal descriptions of computer languages.
- Apply finite automata theory to build recognizers (lexers) for regular languages.
- Apply pushdown automata theory to build recognizers (parsers) for context-free languages.
- Evaluate the role of static analysis in automated program translation.
- Apply tree traversals to convert a syntax tree to low-level code.
- Discuss the limitations that an architecture or execution environment places on the generation of machine code.
- Describe common optimizations and evaluate the tradeoffs associated with good optimization.
Semester-long project

• Compiler for "Decaf" language
  - Implementation in C11 w/ Makefile and integrated test suite
  - Compiles Decaf programs to ILOC & Y86
  - Five major projects: "pieces" of the full system
  - Primary grade based on functionality tiers (like in 261)

• Submission: code (90%) + review (8%) + response (2%)
  - Code can be written in teams of two
    • Benefits vs. costs of working in a team
  - Individual graded code reviews due a week later
  - Review responses (how did your reviewer do?)
Course format

- **Website:** [https://w3.cs.jmu.edu/lam2mo/cs432/](https://w3.cs.jmu.edu/lam2mo/cs432/)
  - Make sure you’re using the right year’s website!

- **Weekly schedule (roughly)**

  - **In-class**
    - Monday: Recap & new topic intro
    - Tuesday: Initial reading & quiz
    - Wednesday: Mini-lecture and discussion
    - Thursday: Detailed reading
    - Friday: In-class lab

  - **Out-of-class**
    - Monday: Project work
    - Tuesday: Project work
    - Wednesday: Project work
    - Thursday: Project work
    - Friday: Project work

- **Formative vs. summative assessment**
  - Formative: quizzes and labs (25% of final grade)
  - Summative: projects and exams (75% of final grade)
Course text(s)

• **Engineering a Compiler, 2\textsuperscript{nd} Edition**
  - Keith Cooper and Linda Torczon
  - 1\textsuperscript{st} chapter scanned; posted under “Files” on Canvas
  - Reserve copy at Rose library

• **Compilers: Principles, Techniques, \& Tools, 2\textsuperscript{nd} Edition**
  - Alfred Aho, Monica Lam, Ravi Sethi, Jeffrey Ullman
  - “The Dragon Book” (premier text on compilers)
  - One section scanned; posted under “Files” on Canvas

• Decaf/ILOC references and type systems reading
  - PDFs on website

• Design patterns reading from GoF book
  - PDF on Canvas
Communication

- Email is always fine (lam2mo)
  - Response likely within a few hours, but no guarantees on weekends or on project deadlines
- Created a Discord server this semester (link in Canvas)
  - Might get a real-time response, maybe not
  - #general, #projects, and #random channels
  - Keep it clean and positive
- Office hours TBD (King Hall 227)
  - In person or virtual – link in Canvas
- Schedule appointments outside of office hours
  - Link in Canvas
Class Policies

- If you test positive for COVID-19 or are consistently coughing and/or sneezing, please stay home
  - Contact me ASAP regarding missed class
  - If you feel a bit ill but well enough to attend class (and are NOT consistently coughing and/or sneezing), please consider wearing a surgical or N95/KN95 mask to protect others
  - Feel free to wear a mask in class or office hours for any reason

- Feel free to bring laptops to class
  - Please do not cause distractions for others

- These policies may change
  - Changes will be announced via Canvas message
Course policies

- Questions?
"The compiler must preserve the meaning of the program being compiled."

- What is a program's meaning?
Compilers encode a program's meaning using an intermediate representation (IR)

- Tree- or graph-based: abstract syntax tree (AST), control flow graph (CFG)
- Linear: register transfer language (RTL), Java bytecode, intermediate language for an optimizing compiler (ILOC)

\[ \begin{align*}
&= \\
&\text{load } b \rightarrow r1 \\
&\text{load } c \rightarrow r2 \\
&\text{add } r1, r2 \rightarrow r3 \\
&\text{load } d \rightarrow r4 \\
&\text{mult } r3, r4 \rightarrow r5 \\
&\text{store } r5 \rightarrow a
\end{align*} \]
Standard compiler framework

- **Front end**: understand the program (src → IR)
- **Back end**: encode in target language (IR → targ)
- **Primary benefit**: easier *re-targeting* to different languages or architectures
Modern compiler framework

- Front-end passes
  - Scanning (lexical analysis)
  - Parsing (syntactic analysis)
- Middle-end passes
  - Static/semantic analysis
  - IR code generation
  - IR optimization
- Back-end passes
  - Instruction selection
  - Machine code optimization
  - Register allocation
  - Instruction scheduling
  - Assembling/linking
- Modern approach: nanopasses
  - Dozens or hundreds of passes (https://llvm.org/docs/Passes.html)
Our Decaf compiler

Source code

int main() {
    int x
    = 4 + 5;
    return x;
}

Tokens

Syntax tree

Checked AST + Symtables

Lexing (P1)

Parsing (P2)

Analysis (P3)

Checked AST + Symtables

IR Code Gen (P4)

Register Allocation (P5)

Machine Code Gen

Run via ILOC interpreter

Run via yas + 261 P4
Compiler rule #2

• The compiler should *help* the programmer in some way
  - What does *help* mean?
Discussion question

• What would be your design goals for a compiler?
  – E.g., what functionality or properties would you like it to have?
  – (Besides rule #1 – correct translation)
Compiler design goals

- Optimize for fast execution
- Minimize memory/energy use
- Catch software defects early
- Provide helpful error messages
- Run quickly
- Be easily extendable
Differing design goals

• What differences might you expect in compilers designed for the following applications?
  – A just-in-time compiler for running server-side user scripts
  – A compiler used in an introductory programming course
  – A compiler used to build scientific computing codes to run on a massively-parallel supercomputer
  – A compiler that targets a number of diverse systems
  – A compiler that targets an embedded sensor network platform

• Optimize for fast execution
• Minimize memory/energy use
• Catch software defects early

• Provide helpful error messages
• Run quickly
• Be easily extendable
Decaf language

- Simple imperative language similar to C or Java
- Example:

```java
// add.decaf - simple addition example

def int add(int x, int y)
{
    return x + y;
}

def int main()
{
    int a;
    a = 3;
    return add(a, 2);
}

$ ./decaf add.decaf
RETURN VALUE = 5
Before Friday

• Readings
  – "Engineering a Compiler" (EAC) Ch. 1 (23 pages)
  – Decaf reference ("Resources" page on website)

• Tasks
  – Complete welcome survey on Canvas
  – Complete first reading quiz on Canvas
  – Write some code in Decaf
  – Test the reference compiler
    • /cs/students/cs432/sp23/decaf
  – Bring your laptop on Friday if you are able
Closing exhortations

• Take care of yourself
  – And if you can, someone else
  – Build (or reconnect with) a support network
  – Protect your boundaries
  – Carve out time to disconnect and rest
  – Talk to someone if things start getting overwhelming

• Have a great semester!