Course Overview

Instructor:
Yizhou Zhang

Teaching Assistant:
Ende Jin
Jianlin Li
Cong Ma

When:
TTh 4pm

Where:
MC 1056

Web page:
https://student.cs.uwaterloo.ca/~cs444

Intro to compilers

Translator b/w representations of program code

- Typically: high-level source code to machine language (object code)
- Not always:
  - Java compiler (javac): Java to interpretable JVM bytecode
  - Java JIT: bytecode to machine code

Do we really need compilers?

- No. Can run programs with interpreter that simulates execution.
- But: best (non-HW) interpreters are at least 10× slower than compiled code
  (e.g., interpreted Python ~30–50× slower than optimized version in C/C++)
  - use up >10× more energy, generate >10× more heat, CO_2
- Run only once ⇒ interpret
- Run many times ⇒ compile

Source code

Optimized for human readability and reasoning

- Expressive, matches human notions of grammar
- Static checking to help avoid programming errors

```
int expr(int n)
{
    int d;
    d = 4 * n * n * (n + 1) * (n + 1);
    return d;
}
```

Assembly and machine code

```
expr:
        push    ebp                 55
        mov     ebp, esp            89 e5
        sub     esp, 4              83 ec 04
        mov     eax, [ebp+8]        8b 45 08
        mov     edx, eax            89 c2
        imul    edx, [ebp+8]        0f af 55
                                    08
        mov     eax, [8+ebp]        8b 45 08
        inc     eax                 40
        imul    edx, eax            0f af d0
        mov     eax, [ebp+8]        8b 45 08
        inc     eax                 40
        imul    eax, edx            0f af c2
        sal     eax, 2              c1 e0 02
        mov     [ebp-4], eax        89 45 fc
        mov     eax, [ebp-4]        8b 45 fc
        leave                       c9
        ret                         c3
```

- Assembler: translate assembly code to machine code

  (In this course: Your compiler will generate assembly code and use an assembler to create machine code)

- CPU: an interpreter
- Information about programmer intent and ability to reason are lost

Unoptimized Code	Optimized Code

```
expr:                              expr:
        push    ebp                        push    ebp
        mov     ebp, esp                   mov     ebp, esp
        sub     esp, 4                     mov     edx, [ebp+8]
        mov     eax, [ebp+8]               mov     edx, eax
        mov     edx, eax                   imul    eax, edx
        imul    edx, [ebp+8]               inc     edx
        mov     eax, [8+ebp]               imul    eax, edx
        inc     eax                        imul    eax, edx
        imul    edx, eax                   sal     eax, 2
        mov     eax, [ebp+8]               leave
        inc     eax                        ret
        imul    eax, edx
        sal     eax, 2
        mov     [ebp-4], eax
        mov     eax, [ebp-4]
        leave
        ret
```

- Optimized code on the right:
  - Fewer instructions
  - Fewer memory accesses
  - Avoids redundant computations

How to translate?

- Source code and machine code mismatch
- Goals:
  - source-level expressiveness
  - best performance of generated code
  - reasonable translation efficiency
    polynomial or even linear time + separate compilation
  - correct, maintainable compiler code

How to translate correctly?

- Correctness is crucial!
  - hard to debug programs with broken compiler...
  - implications for development cost, security
  - non-trivial: PLs are expressive
  - This course: "best practices" that help you build correct compilers
- PLs describe computation precisely
  - PL semantics: mathematical definitions of PLs
  - Translation can be mathematically described
  - Correctness of compilers can be mathematically formulated

  Some compilers have been **proven** to generate correct code!
  (Xavier Leroy, Formal Certification of a Compiler Back End, POPL'06)

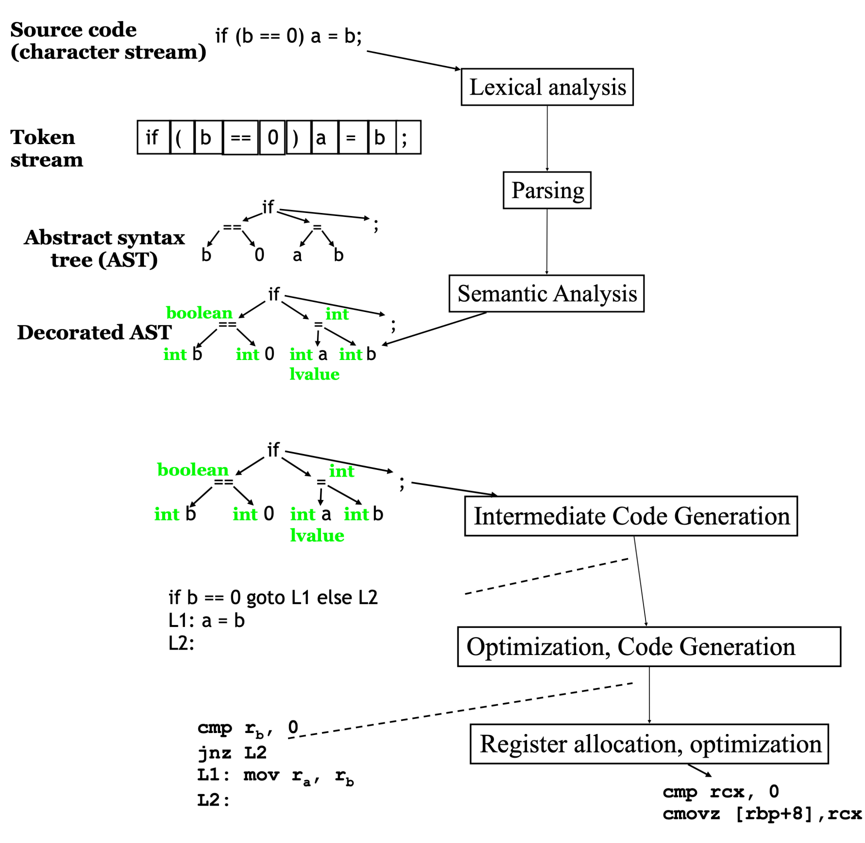
- Looking to learn PL semantics:
  - This course: a little semantics
  - more in CS 442, CS 747

How to translate effectively?

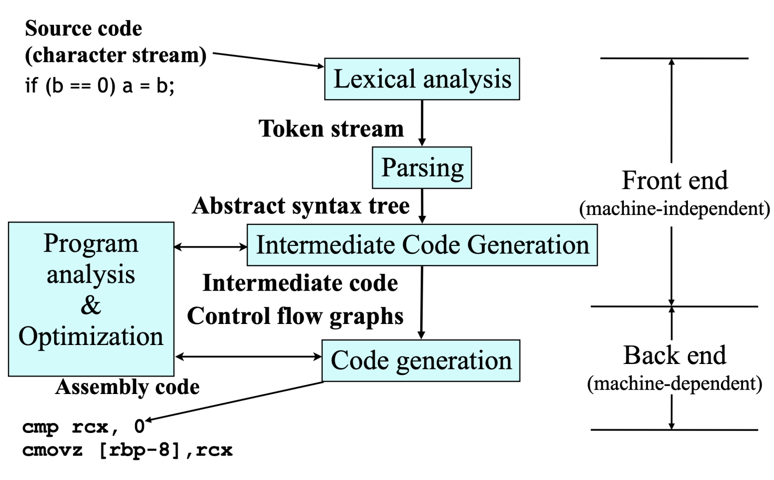
Idea: small easy pieces

- Compiler translates via a series of different **program representations**
- Representations designed to support the necessary program manipulations:
  - type checking
  - optimization
  - code generation

Big picture

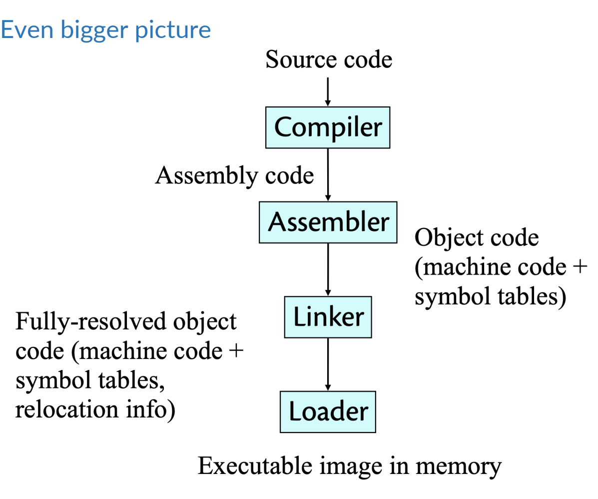


Bigger picture



multiple frontends; multiple backends

Even bigger picture



More administration

Schedule and Assignments

https://student.cs.uwaterloo.ca/~cs444/schedule/

Date	Lecture Topics	Lecture Notes	Upcoming Due Dates
Jan. 7	course overview		
Jan. 9	lexing		
Jan. 14	parsing		Assignment 0 due Jan. 15
Jan. 16	parsing		
Jan. 21	*Instructor away*		
Jan. 23	*Instructor away*		
Jan. 28	parsing		
Jan. 30	name resolution		Assignment 1 due Feb. 2
Feb. 4	name resolution		
Feb. 6	type checking		
Feb. 11	type checking		
Feb. 13	dataflow analysis		Assignment 2 due Feb. 14 (no late penalty through Feb. 16)
Feb. 18	*Reading Week*		
Feb. 20	*Reading Week*		
Feb. 25	dataflow analysis		CS 644 proposal due Feb. 26
Feb. 27	dataflow analysis		Assignment 3 due Mar. 2
Mar. 4	dataflow analysis		
Mar. 6	generating intermediate code		
Mar. 11	generating intermediate code		
Mar. 13	generating intermediate code		Assignment 4 due Mar. 16
Mar. 18	x86 and instruction selecion		
Mar. 20	x86 and instruction selecion		
Mar. 25	compiling OO languages		
Mar. 27	compiling OO languages		Assignment 5 due Mar. 30
Apr. 1	register allocation		
Apr. 3	advanced topics		CS 644 survey due Apr. 6
			Assignment 6 due Apr. 13

Lexing, Parsing, AST (A1)

Name disambiguation and hierarchy checking (A2)
Type checking (A3)
Other static checking (A4)

Fast-paced, frequent deadlines

Intermediate-representation generation, assembly-code generation (A5, A6)

A5 has heaviest weight

A1–A4: Expect to teach yourself the Java Language Specification (JLS).
A5–A6: Expect to teach yourself x86 assembly.

CS 644

Additional work required (15%)
https://student.cs.uwaterloo.ca/~cs444/644/644req

Feb 26: proposal due
April 6: final write-up due

Textbook (not required)

- Tiger book.
  Modern Compiler Implementation in Java (2nd Edition).
  Andrew Appel.

Marking

- 6 assignments to be submitted to Marmoset (68%+8%+10%)
  - Group work
  - public tests: 68%
  - two reports: 8%
    https://student.cs.uwaterloo.ca/~cs444/assignments/report-guidelines
  - Marmoset secret tests: 10%
- End-of-term quiz, part of A6 (14%)
  - Individual work
  - Completed online
  - Timed
  - Covers lecture material

Academic Integrity

- Taken seriously
- Do your own (or your own group's) work
- Reports should document sources you consulted or discussed the project with.

Assignment 0: Due next Wednesday, January 15

https://student.cs.uwaterloo.ca/~cs444/assignments/a0/

- Choose a group of 3 or 4 people
  - Same mark for all (ordinarily)
  - Same group for all assignments (ordinarily)
  - Your experience and success in this course will depend highly on if you work and plan well with your group.
  - Use Piazza to find group members with a matching work style
- Choose an implementation language
  - Some reasonable choices:
    Java, OCaml, Kotlin, Scala, Haskell, Rust, Go, C++, TypeScript
    (common characteristic: statically typed)
  - a reference IR *written in Java* will be provided to you
  - Choose a language you are familiar with.
    (using the course project as an excuse to learn Rust is a decision you may regret)
- Complete the online form
- Will get Marmoset account for group
- Revision control:
  - https://git.uwaterloo.ca
  - Do not post your code to publicly readable repos

Late Policy

For code submissions, the following late policy applies:
    0.5 × best-on-time + 0.5 × best-overall

For report submissions: no late submissions accepted.

All deadlines at 11:59pm (Waterloo time)

Final deadline: April 13.
We will run secret tests on your A6 submission after this date.

Expected learning results

- Deeper understanding of what code is
- A little PL theory
- Algorithms, data structures
- Java, Intel x86
- How to be a good programmer (large code bases, working in a group)

Hopefully, you will be proud of what you achieve in this course!