CS152 Compiler Design
Instructor and TA

• Rajiv Gupta  Professor @CSE
  Office hours: By Appointment
  Email: gupta@cs.ucr.edu

• Mahbod Afarin
  TA (Lab Session)
  Office hours: By Appointment
  Email: mafar001@ucr.edu
Administrivia

• Materials: lectures, syllabus and schedule, policies
  - My website: 
    - [https://www.cs.ucr.edu/~gupta/teaching/152-21s/](https://www.cs.ucr.edu/~gupta/teaching/152-21s/)

• iLearn
  - Grading
  - Announcements

• For questions
  - Email, appointments via zoom
  - about lectures → Contact Me
  - about project → Contact TA
Textbook (primary)

- Compiler Construction – Principles and Practice 1997 by Kenneth Louden
  - 1 copy on reserve in library
  - TAs have copies also
Textbook (references)
Project and Lab

• Three phases (4/2; 4/16; 5/14—6/4)
  - Roughly three weeks each
  - 10% + 12% + 13% = 35%

• Build a compiler frontend with tools
  - introduced by TAs in Lab Sessions

• Team of two / individual
Grading

• Three parts
  - 35% : Project
  - 30% : Exam I  4/30/2021, 3:00-3:50pm
  - 35% : Exam II  6/08/2021, 7:00-10:00pm
Academic integrity

• What constitutes academic dishonesty?
  - Cheating, fabrication, plagiarism, unauthorized collaboration (or facilitating any of these)

• What are the penalties and sanctions?
  - receiving an F for the class and filing a report of the incident
  - Ignorance is no excuse
Chapter 1

What is a Compiler
What is a Compiler?

- A program that translates a program written in one language into a program written in another language.

- A Interpreter is a program that reads a program and produces the results of executing that program.
Language vs. Compiler

- **C/C++** programs are typically compiled
- **Script** (JS/Python) programs were typically interpreted-only
- **Java** programs are compiled to bytecode (code for JVM)
  - then interpreted or (just-in-time) compiled
Compilation vs. Interpretation

- **Advantages** (Interpretation)
  - support dynamic features (dynamic typing & scoping)
  - portability

- **Disadvantages** (Interpretation)
  - slower
  - less reliable
Compilation Phases
Compilation Phases

• **Phases**
  - A typical compiler is organized into phases
    - Phases I-IV: **Frontend**
    - Phases V-VI: **Backend**

• **Passes**
  - A traversal of the whole code representation
Compilation Phases

- Example
  - “Journey” of a statement

\[ a[i] = 4 + 2 \]
Compilation Phases

• Lexical Analysis
  - input: source code (character stream)
  - output: token stream & errors

\[
a[i] = 4 + 2
\]

Emma likes cats

identifiers:
- a
- i

tokens:
- "Emma"
- "likes"
- "cats"

numbers:
- 4
- 2

operators:
- +

This is a diagram showing the compilation phases:

1. Lexical Analysis
2. Syntax Analysis
3. Semantics Analysis
6. Target Code Gen.
Compilation Phases

- Syntax Analysis
  - input: token stream
  - output: parse tree & errors
Compilation Phases

- **Syntax Analysis**
  - input: *token stream*
  - output: *abstract syntax tree & errors*
Compilation Phases

- Semantics Analysis
  - input: syntax tree
  - output: annotated syntax tree & errors

```
a[i] = 4 + 2
```
Compilation Phases

• Intermediate Code (IR) Gen.
  - input: annotated syntax tree
  - output: IR (three-address code)

\[ a[i] = 4 + 2 \]

```
a[i] = 4 + 2
```

```
t = 4 + 2
```

```
a[i] = t
```
Compilation Phases

• Code Optimizations (many times)
  - input: IR
  - output: optimized IR

```
t = 4 + 2
a[i] = t

\[
t = 6
a[i] = t
\]

\[
a[i] = 6
\]
Compilation Phases

- **Target Code Gen.**
  - input: IR
  - output: assembly/machine code

```plaintext
a[i] = 6
```

```plaintext
mov R0, i       ;; value of i → R0
mul R0, 4       ;; multiply R0 by 4
mov R1, &a      ;; addr of a → R1
add R1, R0      ;; add R0 to R1
mov *R1, 6      ;; 6 → addr in R1
```
Compilation Phases

• Target Code Gen.
  - input: **IR**
  - output: **assembly/machine code**

```assembly
mov R0, i  ;; value of i → R0
mul R0, 4  ;; multiply R0 by 4
mov R1, &a ;; addr of a → R1
add R1, R0 ;; add R0 to R1
mov *R1, 6 ;; 6 → addr in R1

mov R0, i  ;; value of i → R0
shl R0, 2  ;; shift left 2 bits
mov &a[R0], 6 ;; 6 → addr in R1
```
Compilation Phases

- Example
  - “Journey” of a statement

\[ a[i] = 4 + 2 \]

```
mov R0, i
shl R0, 2
mov &a[R0], 6
```