CS152 Compiler Design
Instructor and TA

• Rajiv Gupta  Professor @CSE
  Office hours: Tuesday & Thursday
  10:00am – 11:00am
  Email: gupta@cs.ucr.edu

• Abbas Mazloumi  TA (Lab Session)
  Office hours: Friday, 2:00pm – 4:00pm
  Email: amazl001@ucr.edu
Administrivia

• Materials: lectures, syllabus and schedule, policies
  - My website: https://www.cs.ucr.edu/~gupta/teaching/152-20summ/

• iLearn
  - Grading
  - Announcements

• For questions
  - Piazza, email, office hours on zoom
  - about lectures → Contact Me
  - about project → Contact TAs
Textbook (primary)

- Compiler Construction – Principles and Practice 1997 by Kenneth Louden

- 1 copy to on reserve in library

- TAs have copies also
Textbook (references)
Project and Lab

• Three phases (6/22; 7/1; 7/13—7/22)
  - 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  7/6/2020, 10:30-11:30am
  - 35% : Exam II  7/24/2020, 8:00-10:00am
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
```

<table>
<thead>
<tr>
<th>Text</th>
<th>Token Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>identifier</td>
</tr>
<tr>
<td>[</td>
<td>left bracket</td>
</tr>
<tr>
<td>i</td>
<td>identifier</td>
</tr>
<tr>
<td>]</td>
<td>right bracket</td>
</tr>
<tr>
<td>=</td>
<td>assignment</td>
</tr>
<tr>
<td>4</td>
<td>number</td>
</tr>
<tr>
<td>+</td>
<td>plus sign</td>
</tr>
<tr>
<td>2</td>
<td>number</td>
</tr>
<tr>
<td>“Emma”</td>
<td>noun</td>
</tr>
<tr>
<td>“likes”</td>
<td>verb</td>
</tr>
<tr>
<td>“cats”</td>
<td>noun</td>
</tr>
</tbody>
</table>
Compilation Phases

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

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

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

```
a    identifier
[    left bracket
i    identifier
]    right bracket
    assignment
=    number
4    number
+    plus sign
2    number
```
Compilation Phases

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

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

```
    assign
   /    |
  add   subscript
   |     |
  4     i
   |     |
 array of integer integer
```

Compilation Phases

  - input: annotated syntax tree
  - output: IR (three-address code)

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

```
4
\text{add}
```

```
i
\text{subscript}
```

```
a
```
array of integer

```
2
```
integer

```
t = 4 + 2
```

```
a[i] = t
```

Diagram:

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

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

\[
\begin{align*}
t &= 4 + 2 \\
a[i] &= t \\
t &= 6 \\
a[i] &= t \\
a[i] &= 6
\end{align*}
\]
Compilation Phases

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

\[ a[i] = 6 \]

\[
\begin{align*}
mov & \ R0, \ i \quad ;; \ value \ of \ i \rightarrow \ R0 \\
mul & \ R0, \ 4 \quad ;; \ multiply \ R0 \ by \ 4 \\
mov & \ R1, \ \&a \quad ;; \ addr \ of \ a \rightarrow \ R1 \\
add & \ R1, \ R0 \quad ;; \ add \ R0 \ to \ R1 \\
mov & \ *R1, \ 6 \quad ;; \ 6 \rightarrow \ addr \ in \ R1
\end{align*}
\]
Compilation Phases

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

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

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

- Example
  - “Journey” of a statement

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

\[
\text{mov R0, i}
\text{shl R0, 2}
\text{mov &a[R0], 6}
\]