UNIVERSITY OF CALIFORNIA
UC RIVERSIDE

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

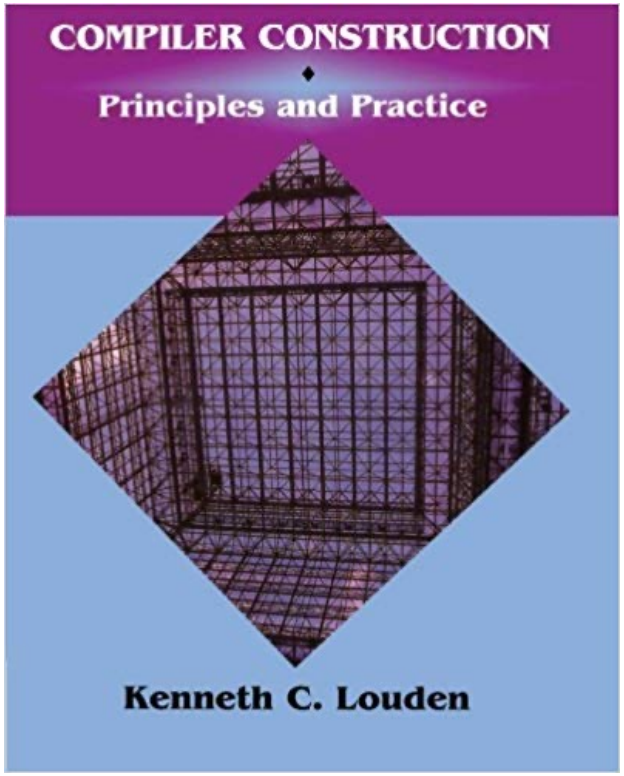
Administrivia

- Materials: lectures, syllabus and schedule, policies
  - My website:
  - https://www.cs.ucr.edu/~gupta/teaching/152-21summ/

- iLearn
  - Power points & Lecture Recordings
  - Grading
  - Announcements

- For questions
  - Email, appointments via zoom
  - about lectures → Contact Me
  - about project → Contact TA

Textbook (primary)

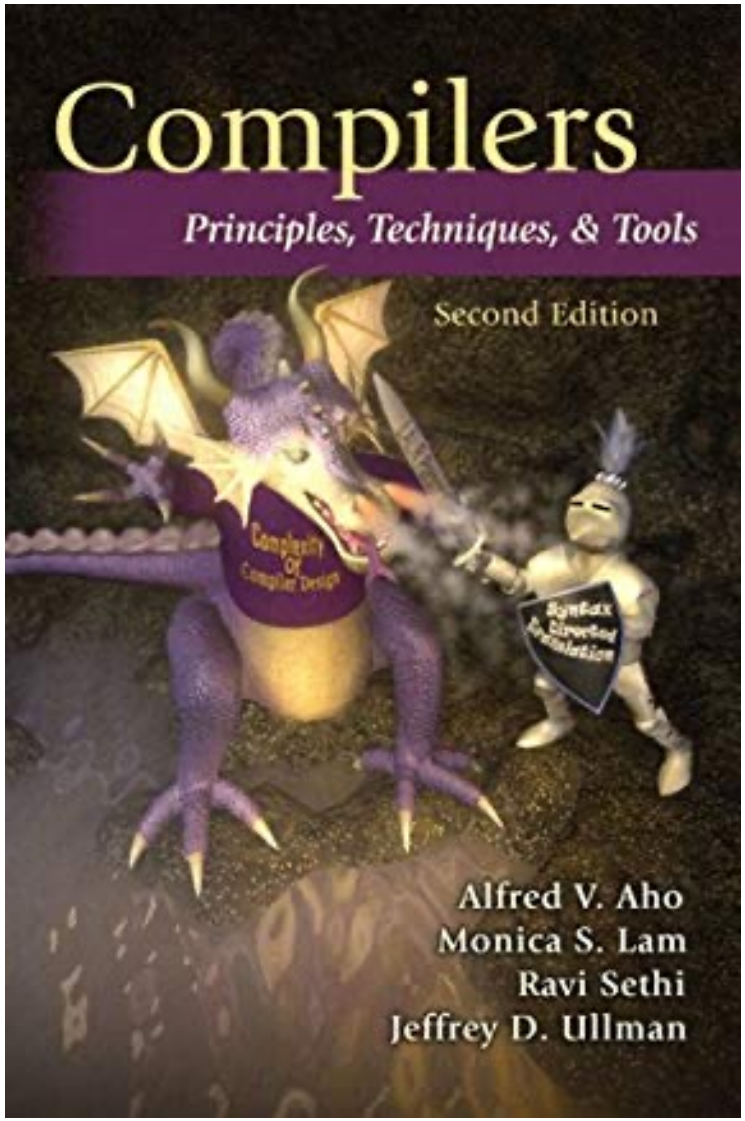


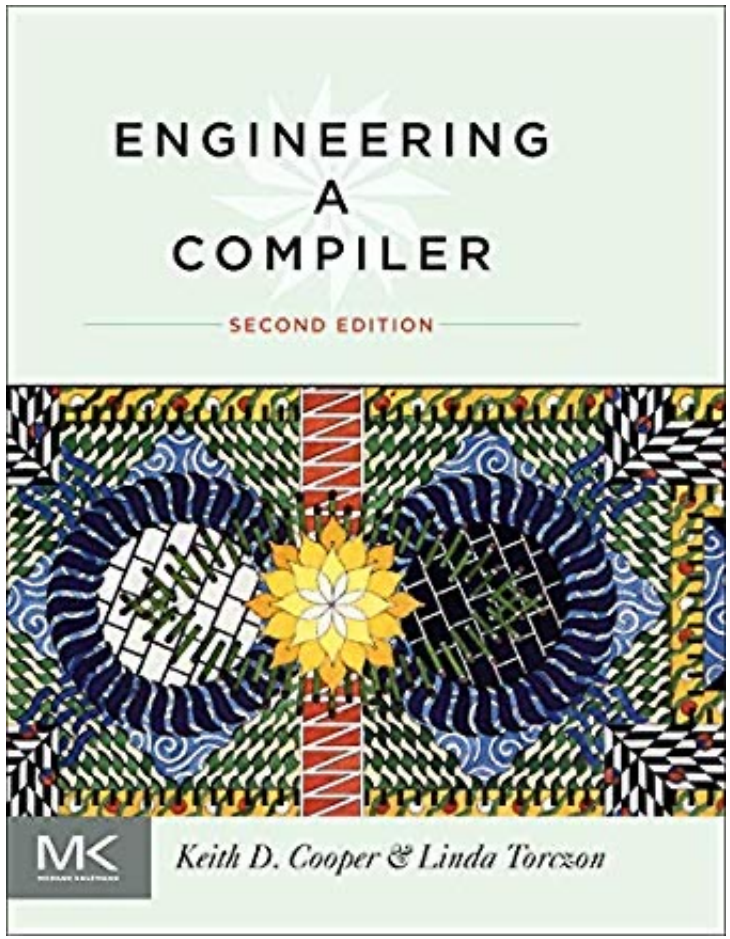
- Compiler Construction – Principles and Practice 1997
  by Kenneth Louden

- ~~1 copy to on reserve in library~~

- ~~TAs have copies also~~

Textbook (references)





Project and Lab

- Three phases (6/21; 6/30; 7/12—7/21)
  - 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 7/07/2021, 9:50-10:50am
  - 35% : Exam II 7/23/2021, 8:00-10:00am

Academic integrity

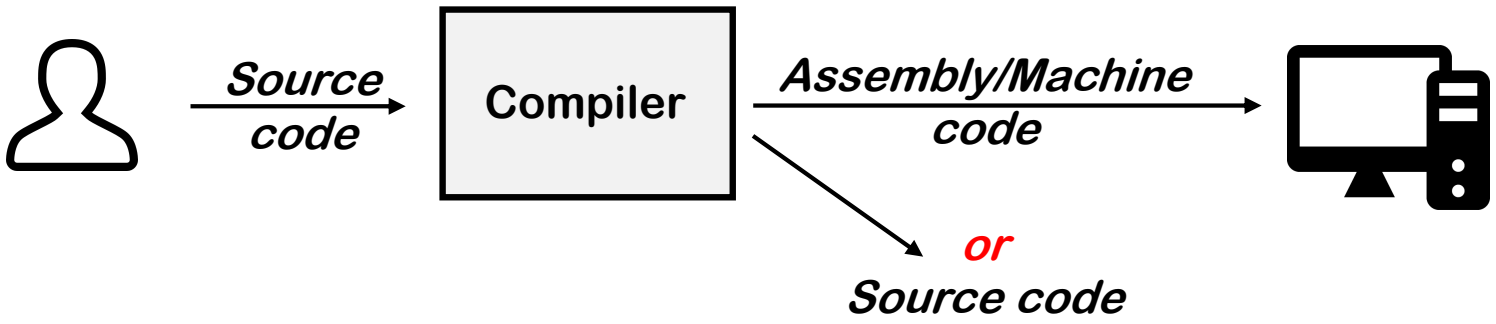
- What constitutes academic dishonesty?
  - Cheating, fabrication, plagiarism, unauthorized collaboration (or facilitating any of these)
- What are the penalties and sanctions?
  - receiving an F for the class and filing a report of the incident
  - Ignorance is no excuse

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.



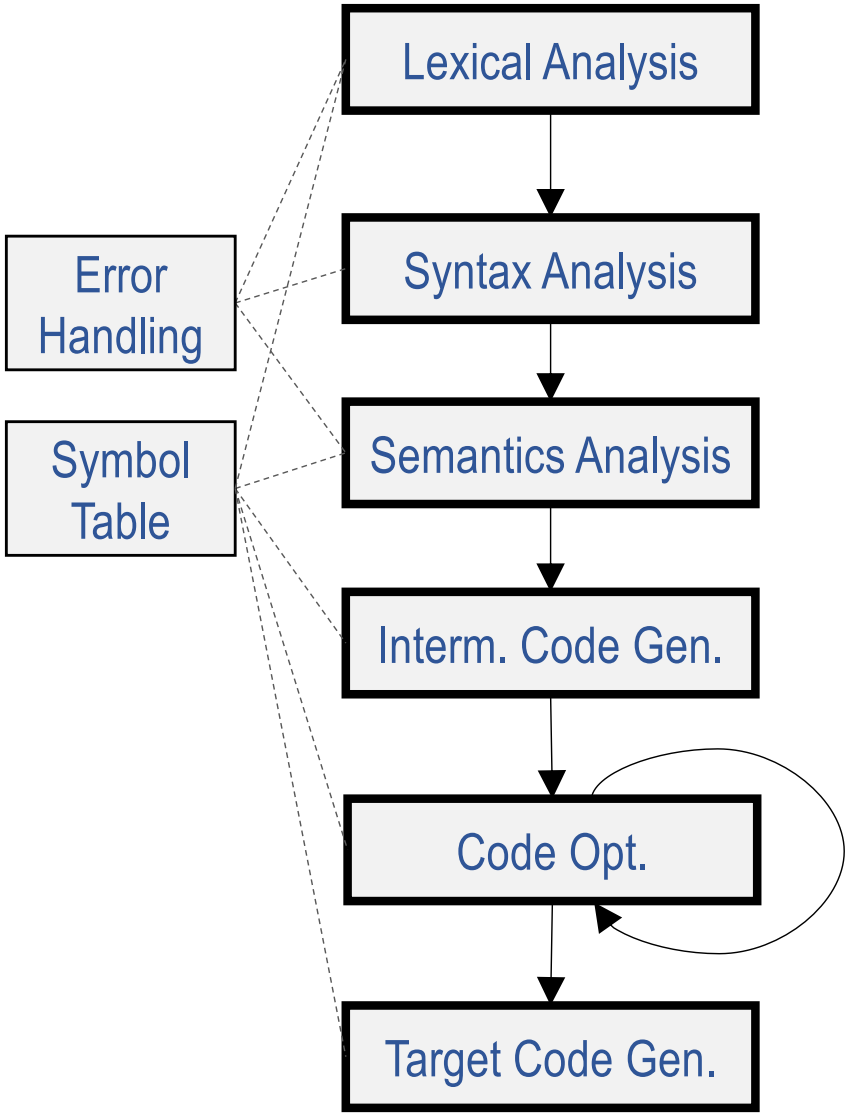
- A **Interpreter** is a program that reads a program and produces the results of executing that program.

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 Phases

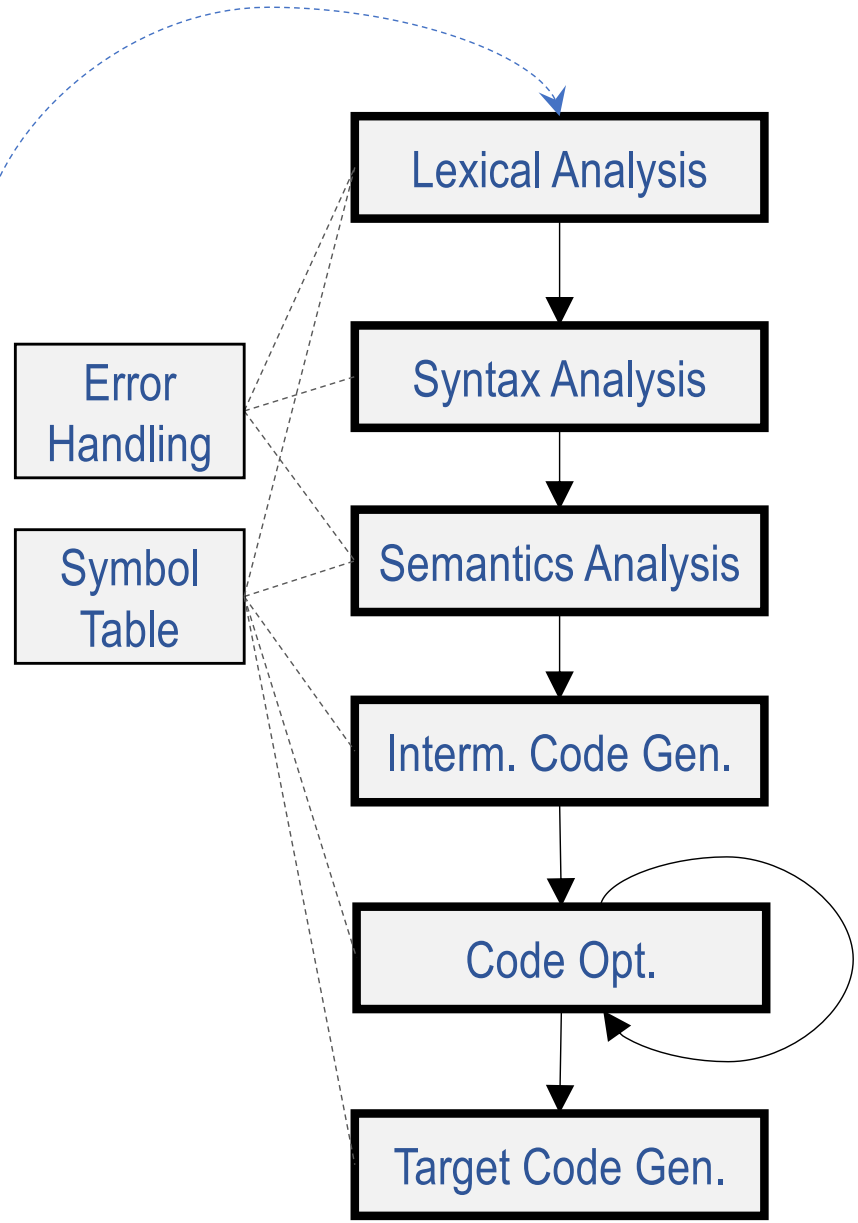
- **Phases**
  - A typical compiler is organized into phases
  - Phases I- IV: Frontend
  - Phases V-VI: Backend
- **Passes**
  - A traversal of the whole code representation



Compilation Phases

- Example
  - "Journey" of a statement

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



Compilation Phases

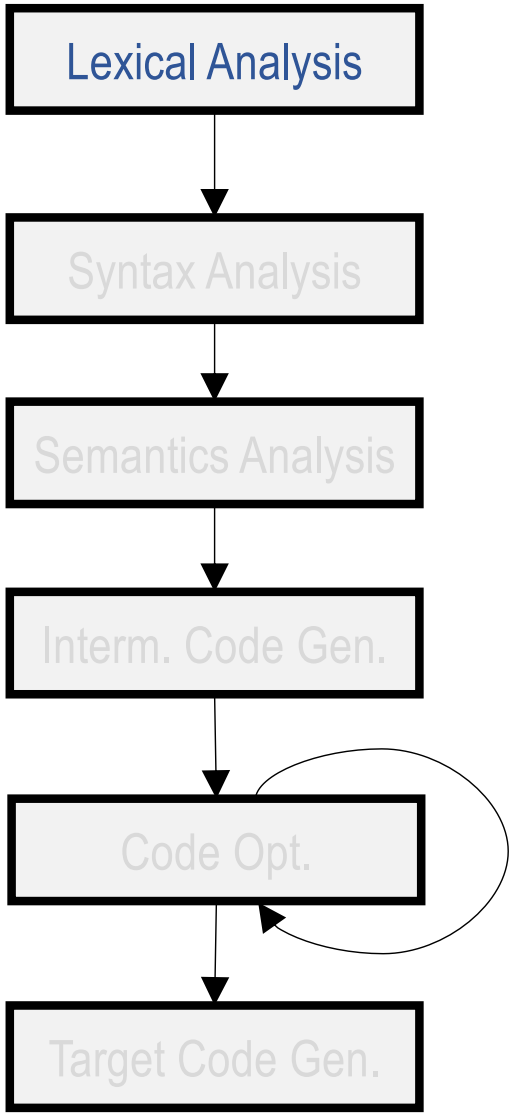
- Lexical Analysis
  - input: source code (character stream)
  - output: token stream & errors

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

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

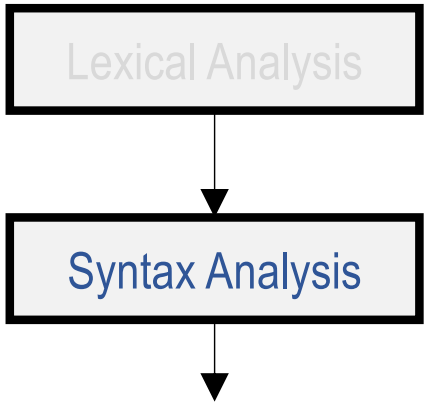
Emma likes cats

"Emma"	noun
"likes"	verb
"cats"	noun

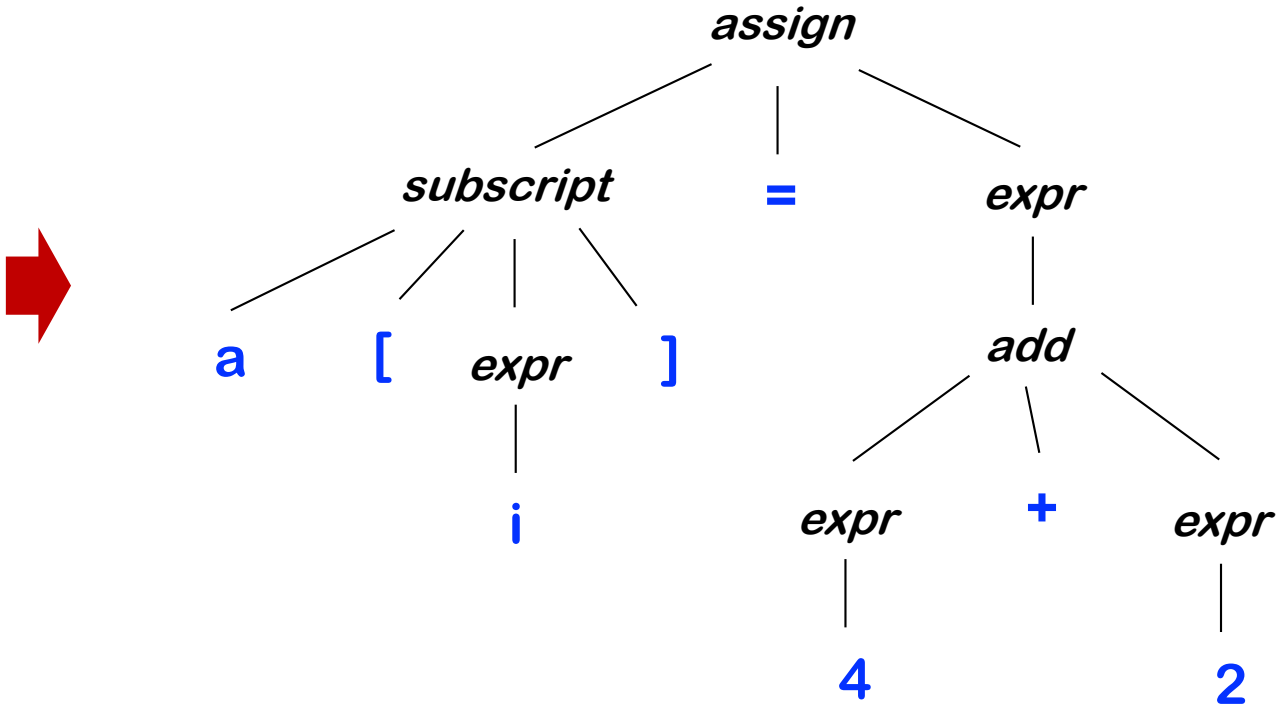


Compilation Phases

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

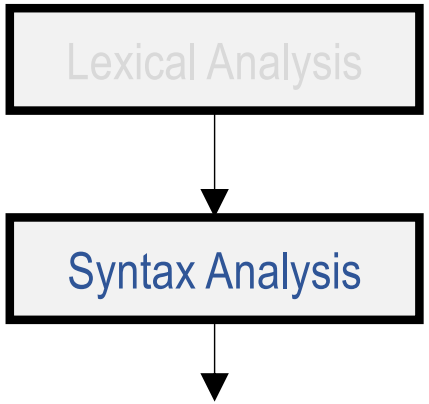


a	identifier
[	left bracket
i	identifier
]	right bracket
=	assignment
4	number
+	plus sign
2	number



Compilation Phases

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

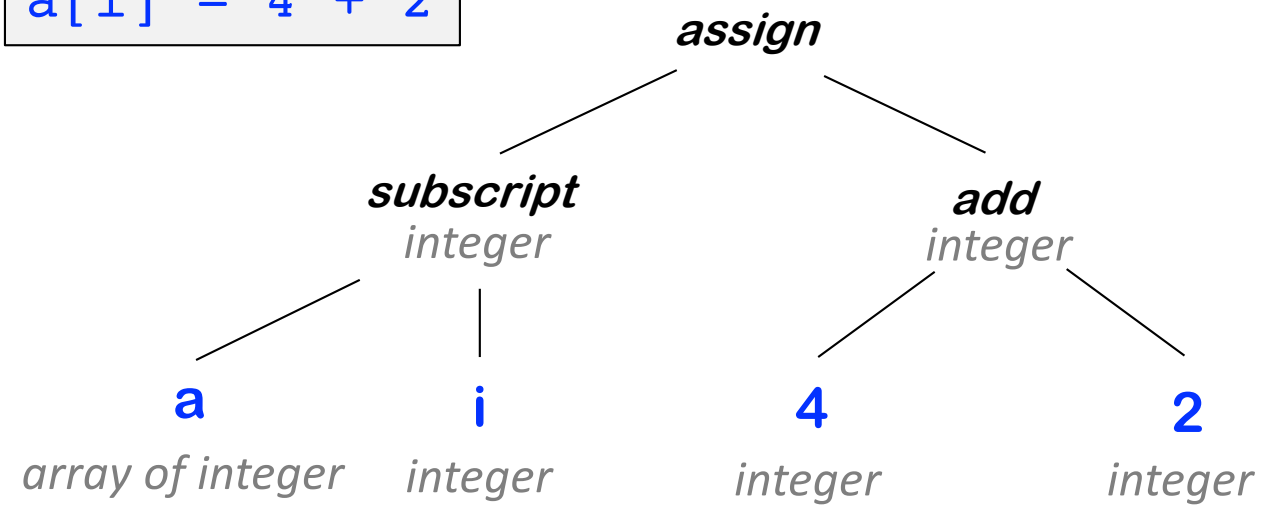


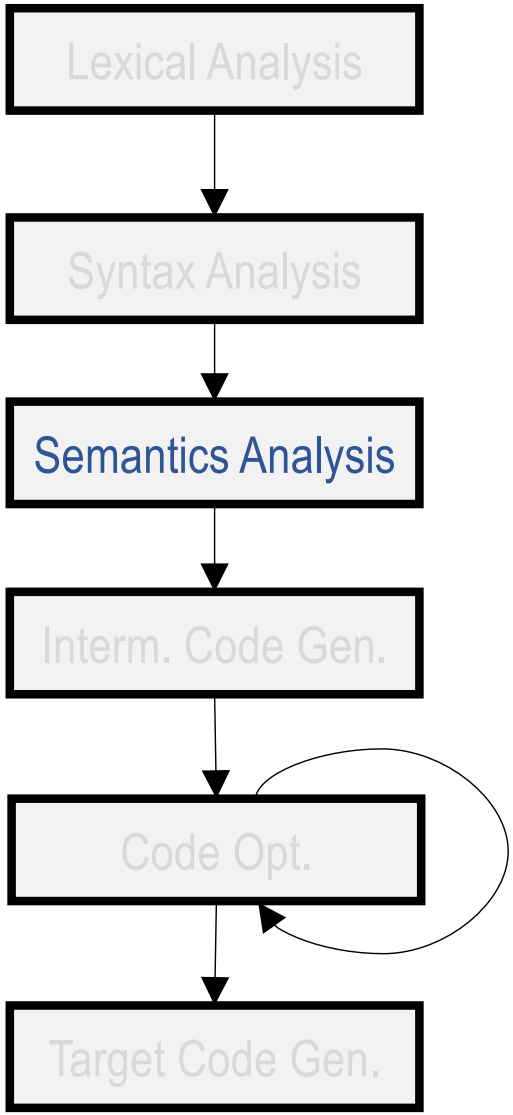
a	identifier
[	left bracket
i	identifier
]	right bracket
=	assignment
4	number
+	plus sign
2	number

Compilation Phases

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

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

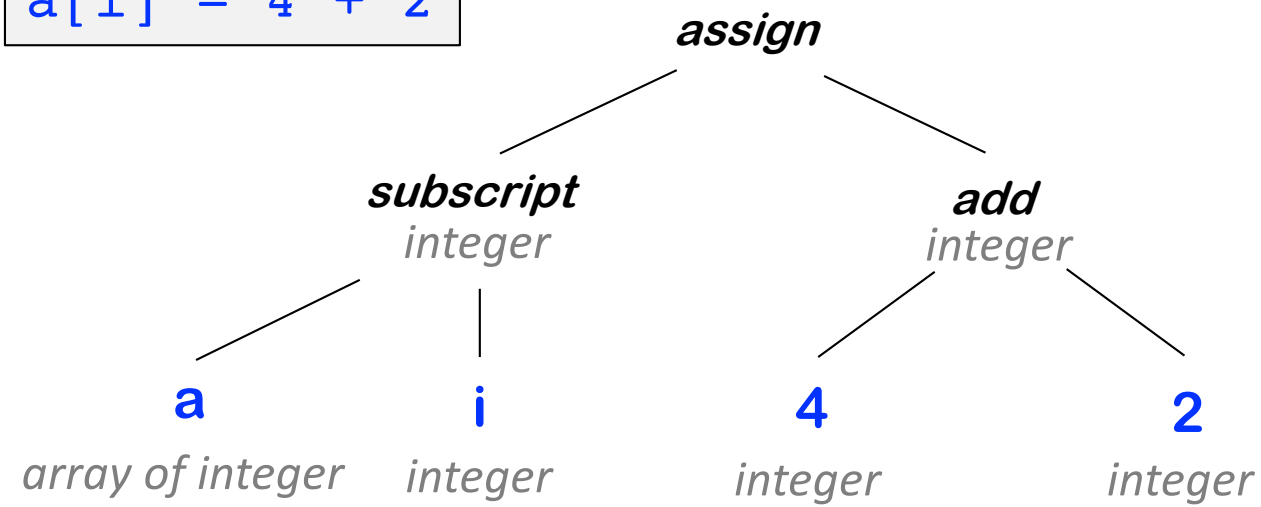




Compilation Phases

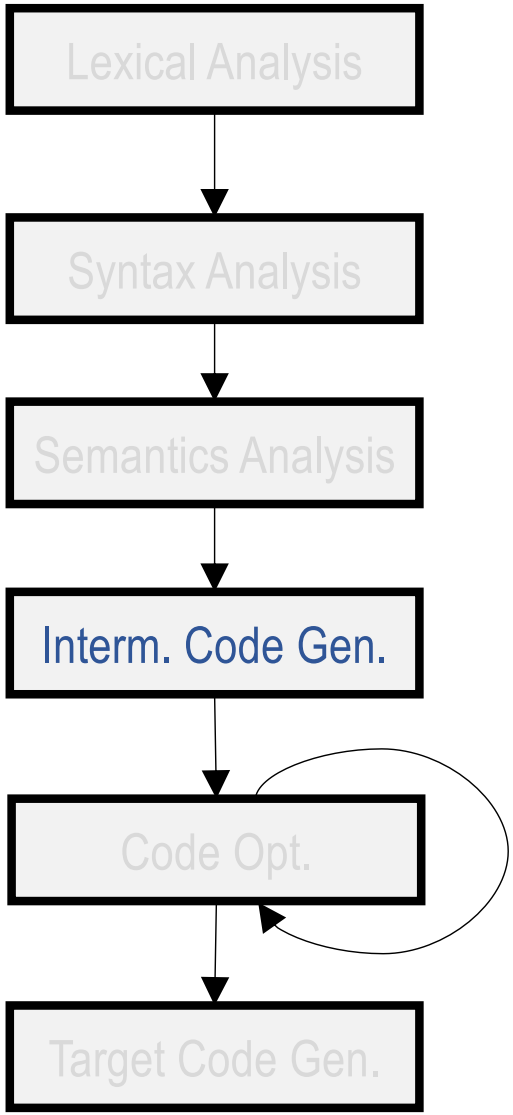
- Intermediate Code (IR) Gen.
  - input: annotated syntax tree
  - output: IR (three-address code)

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



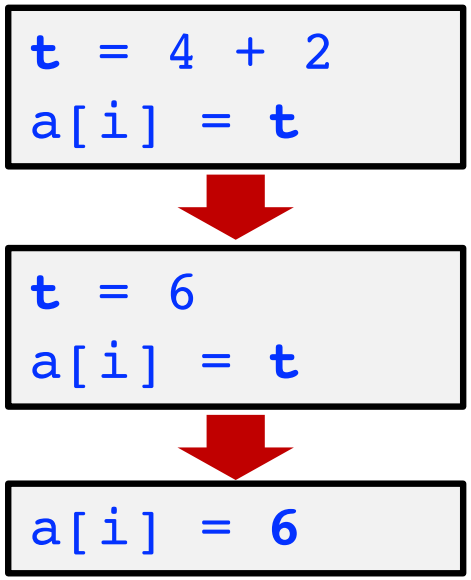
temp. var.

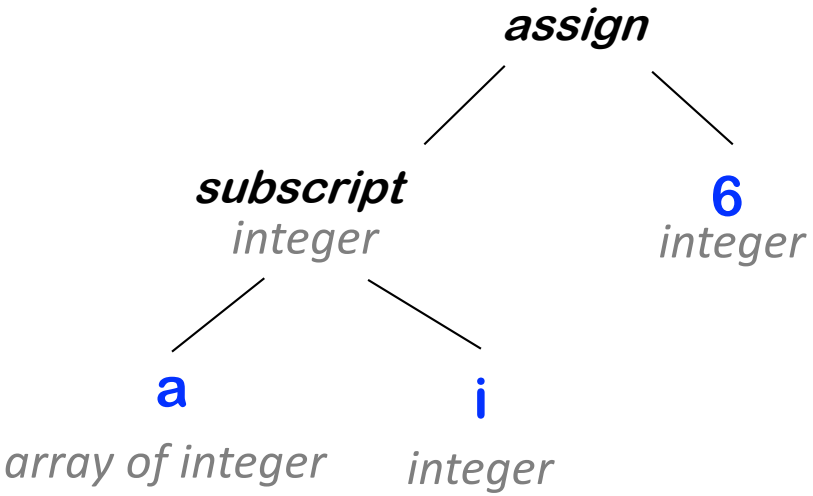
```
t = 4 + 2
a[i] = t
```

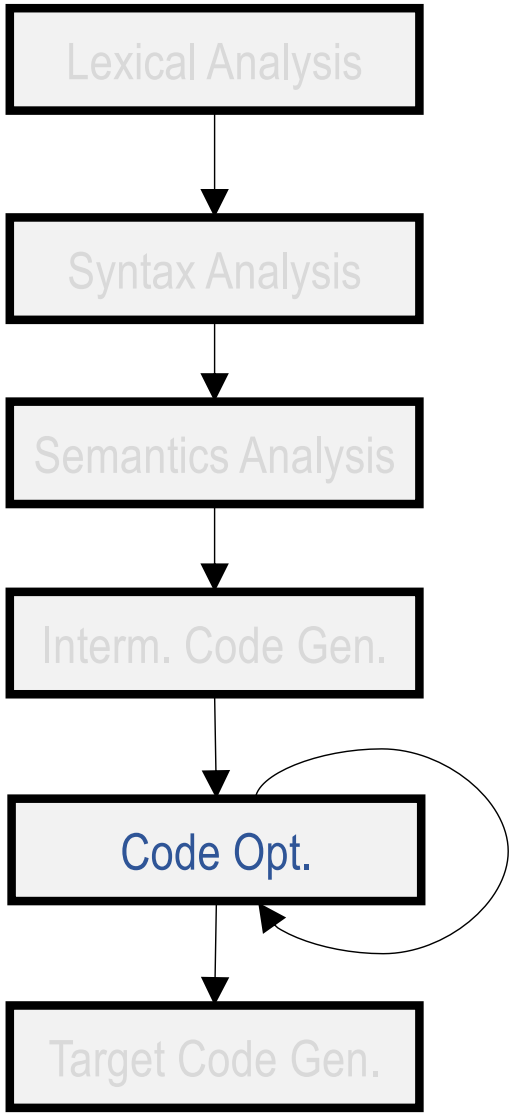


Compilation Phases

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





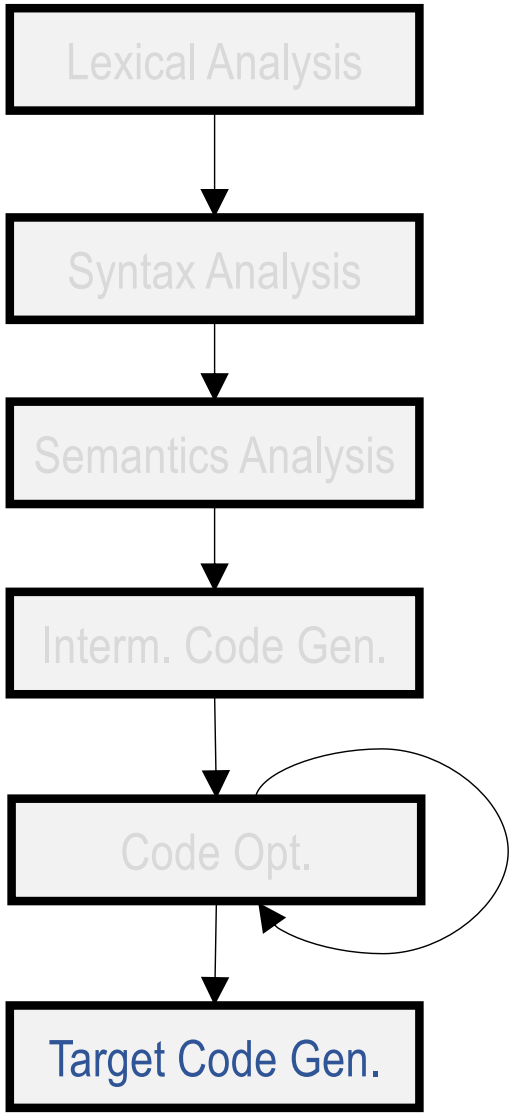


Compilation Phases

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

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

```
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
```

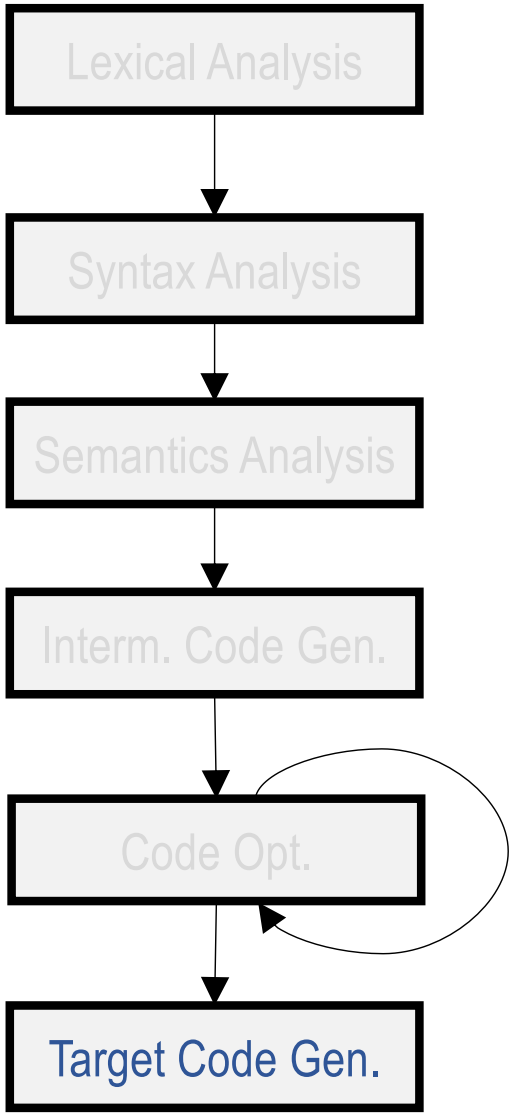


Compilation Phases

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

```
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
```

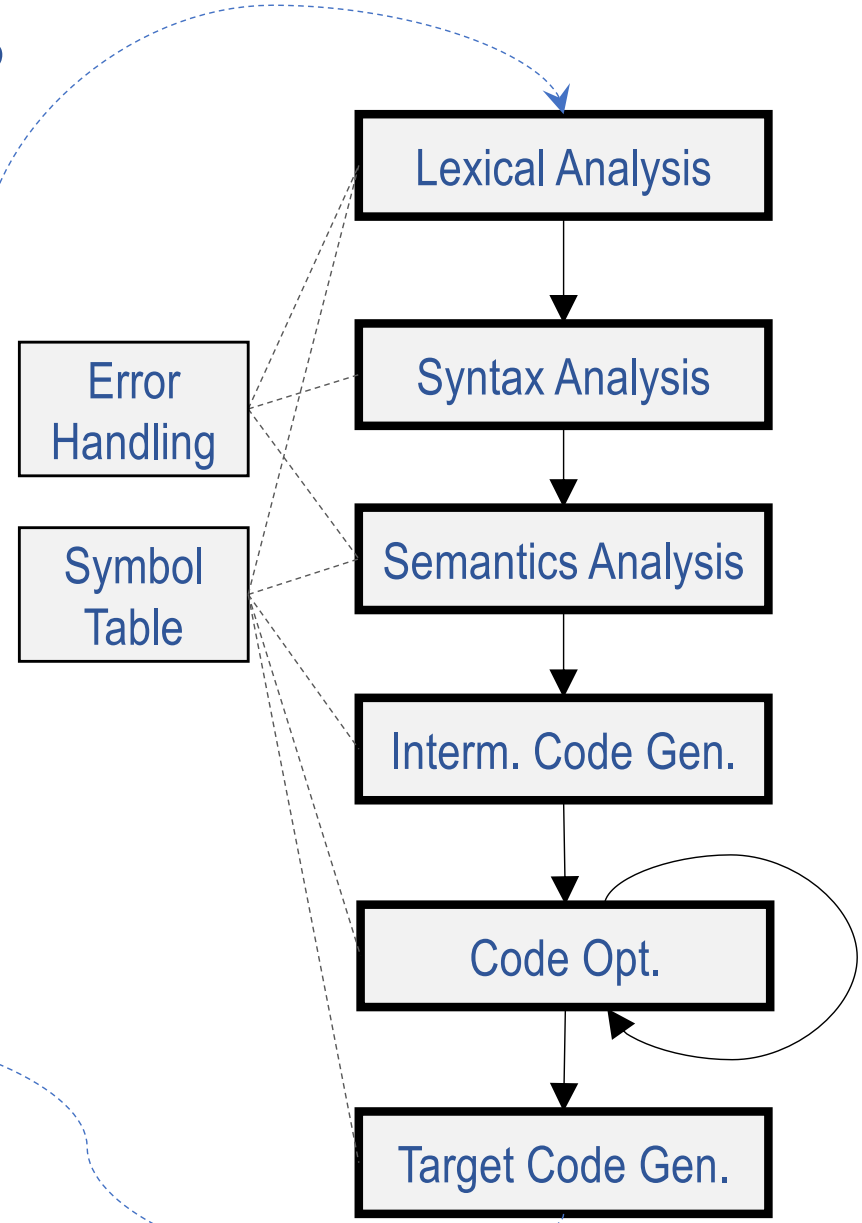
```
mov R0, i   ;; value of i → R0
shl R0, 2   ;; shift left 2 bits
mov &a[R0], 6 ;; 6 → addr in R1
```



Compilation Phases

- Example
  - "Journey" of a statement

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




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