Introduction to Compiler Construction

Compilation: Preliminaries
Outline

1. Compilers
2. Interpreters
3. Phases of Compilation
4. Why Study Compilers?
Compilers

A compiler translates a source language program into a target language program:

- **source language program** (high-level)
- **compiler**
- **target language program** (low-level)

Examples of source language: C, Java, j--, iota

Examples of target language: Intel x86 instructions, JVM instructions, Marvin Machine instructions
A compiler translates a source language program into a target language program:

Examples of source language: C, Java, $j-$, $iota$

Examples of target language: Intel x86 instructions, JVM instructions, Marvin Machine instructions
A programming language specification consists of:

• Syntax of tokens
• Syntax of constructs such as classes, methods, statements, and expressions
• Semantics (ie, meaning) of the constructs
A programming language specification consists of:

- Syntax of tokens
- Syntax of constructs such as classes, methods, statements, and expressions
- Semantics (i.e., meaning) of the constructs
A machine's instruction set along with its behavior is referred to as its architecture.

Examples:

- Intel x86
- Java Virtual Machine (JVM)
- Marvin Machine
A machine's instruction set along with its behavior is referred to as its architecture.

Examples:
- Intel x86
- Java Virtual Machine (JVM)
- Marvin Machine
Interpreters

An interpreter executes a source language program directly:

Examples of interpreters: Bash, Python
An interpreter executes a source language program directly:

Examples of interpreters: Bash, Python
Phases of Compilation

A compiler can be broken into a front end and a back end:

- **Source language program**
- **Intermediate Representation (IR)**
- **Front end**
- **Back end**
- **Target language program**

Separating the front end from the back end enables code re-use.
Phases of Compilation

A compiler can be broken into a front end and a back end:

![Compilation Diagram]

† Intermediate Representation

Separating the front end from the back end enables code re-use
Phases of Compilation

The front end can be decomposed into a sequence of analysis phases:

- Source language program
- Scanner
- Tokens
- Parser
- Abstract Syntax Tree (AST)
- Semantics
- Intermediate Representation (IR)
The front end can be decomposed into a sequence of analysis phases:

```
source language program ➔ scanner ➔ tokens ➔ parser ➔ AST† ➔ semantics ➔ IR
```

† Abstract Syntax Tree
Phases of Compilation

The back end can be decomposed into a sequence of synthesis phases:
The back end can be decomposed into a sequence of synthesis phases:
Phases of Compilation

A compiler sometimes has an optimizer between the front end and the back end.
Phases of Compilation

A compiler sometimes has an optimizer between the front end and the back end:
Why Study Compilers?

Compilers are larger programs than the ones you have written so far. Compilers make use of things you have learned about earlier. You learn a lot about the source language and the target machine. Compilers are still being written for new languages and targeted to new architectures. There is a good mix of theory and practice. Compiler writing is a case study in software engineering. Compilers are programs and writing programs is fun.
Why Study Compilers?

Compilers are larger programs than the ones you have written so far

Compilers make use of things you have learned about earlier

You learn a lot about the source language and the target machine

Compilers are still being written for new languages and targeted to new architectures

There is a good mix of theory and practice

Compiler writing is a case study in software engineering

Compilers are programs and writing programs is fun