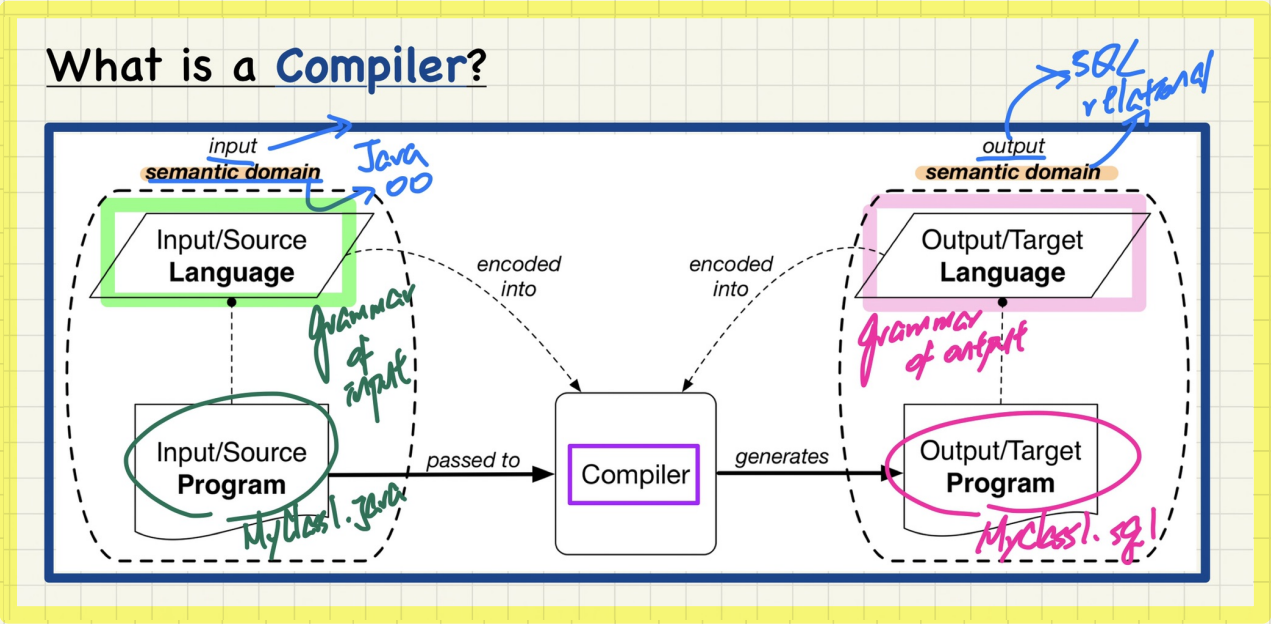
Lecture 2 - Sep. 13

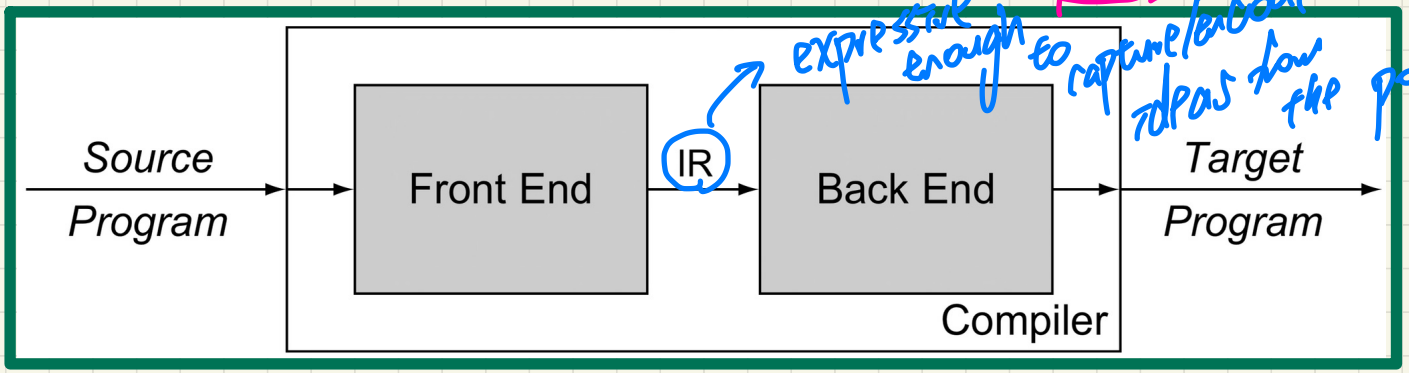
Overview of Compilation

Components of a Compiler:
Frontend, Optimizer, Backend
Introducing Scanner

- Survey on Programming Test Time
- Office Hours



Compiler: Typical Infrastructure (1)



Concrete Syntax vs. Abstract Syntax

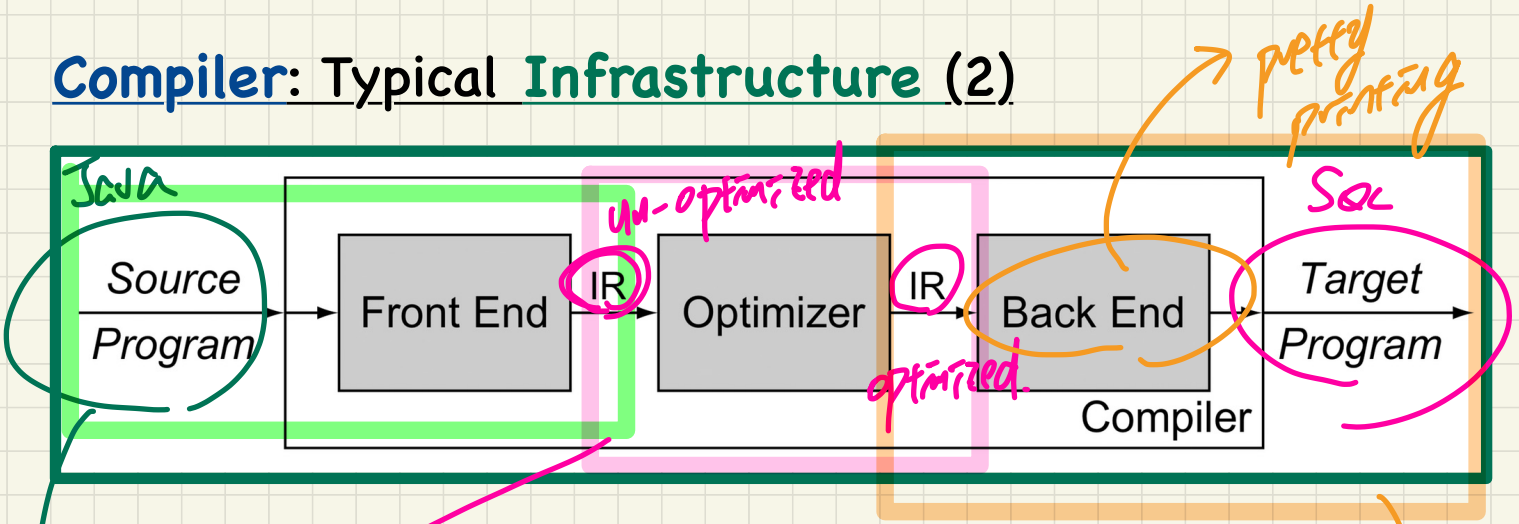
Q. How many IRs are necessary to build a number of compiler?

- Java-to-C
- C#-to-C
- Java-to-Python
- C#-to-Python

IR1: Java-to-machine

IR2: machine-to-Java

Compiler: Typical Infrastructure (2)



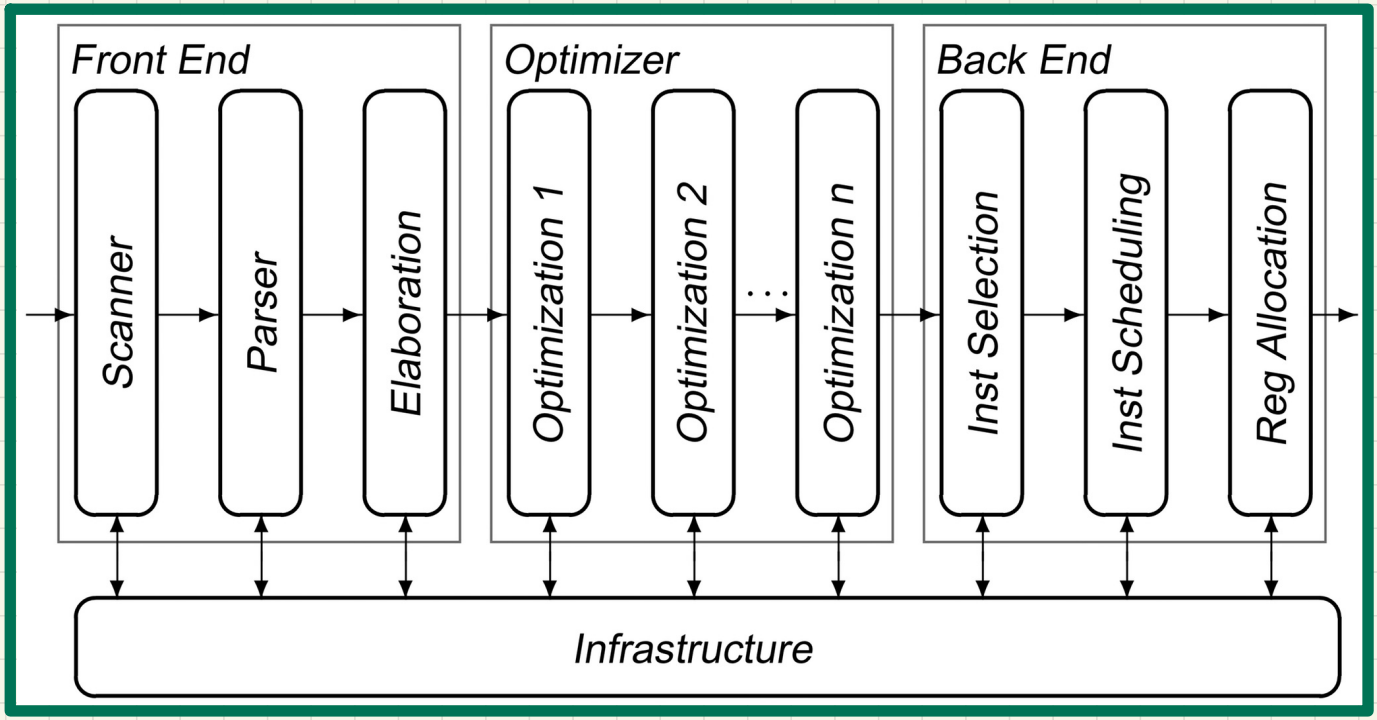
Q. What does the behaviour of the target program depend upon?

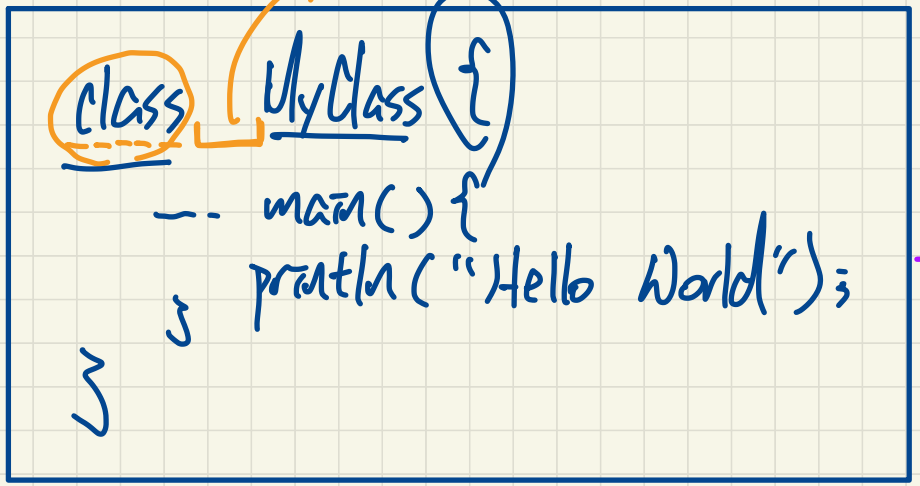
1. input accurately encoded in IR

2. un-optimized IR accurately encoded in optimized IR

3. optimized IR accurately encoded in output

Example Compiler 1: Infrastructure



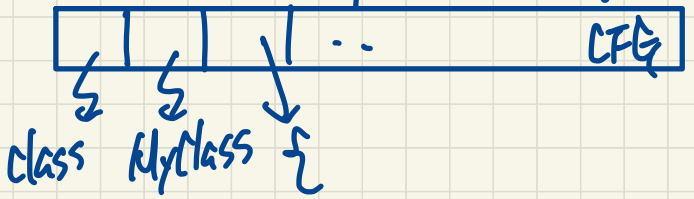


- A parse tree means the input is syntactically correct.
- A parse tree does not necessarily have a well-defined meaning.

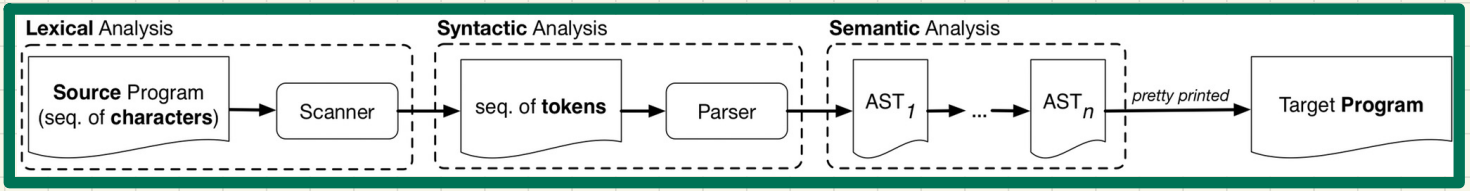
lexical (scanner) → output: token

"class" → token → keywords, identifier, ...

syntactic (parser) → parse tree w.r.t CFG



Compiler Infrastructure: Scanner, Parser, Optimizer



Analogy: Compare Compilation to Essay Writing

Introduction

Contemporary technologies in today's information society are not merely an institutional system, instead, they are a system of material objects designed by those who intend to exercise the social requirements and their hegemonic purposes: command, control, and exploitation. In this essay, one main thesis – contemporary technologies are not neutral – will be revealed by first looking at how Feenberg's notions of dialectical technological rationality and technical code provide a generic template for explaining how technologies can combine the social and political requirements under a particular capitalist social context, and then examining two different standings on arguing the "un-neutrality" of technologies: While Margolis and Resnick argue for the ethical ideas, Winner, Goodman, McDermott, and Robins and Webster argue against the blamable messages embedded within technologies.

Summaries of Arguments from Sources

In his work, Cressman (2004) describes how Feenberg develops his notions of dialectical technological rationality and his concept of the technical code based on Marx's technological ambivalence and Marcuse's technological rationality. Feenberg's technical code can be defined as the general rule of integrating social requirements and the technical advancement into a single technological artifact, which frequently binds technological applications to hegemonic purposes (Cressman 2004). Based on Marx's notion of "design critique" of technology, Feenberg claims that the contemporary social system of capitalism has shaped the sort of technology we are using and even guides what we will have in the future. A capitalist system mainly requires the control over the majority of the working class, and hence division of the labour force is implemented, and

- words → lexical (spellings)
- sentences → syntactic (grammar)
- meaning

I tennis.

fails parser.

while-Loop: Context-Free Grammar (CFG)

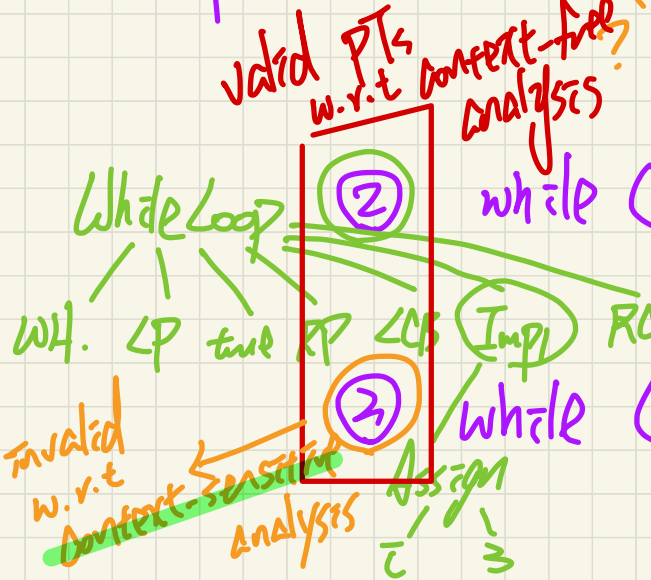
$123 ? a234

WhileLoop	::=	`WHILE` `LPAREN` *BoolExpr* `RPAREN` `LCBRAC` *Impl* `RCBRAC`
Impl	::=	
	|	*Instruction* `SEMICOL` *Impl*

Input: ① while true { print(...); }

valid PTs w.r.t context-free analysis

⇒ parse error (no parse tree)



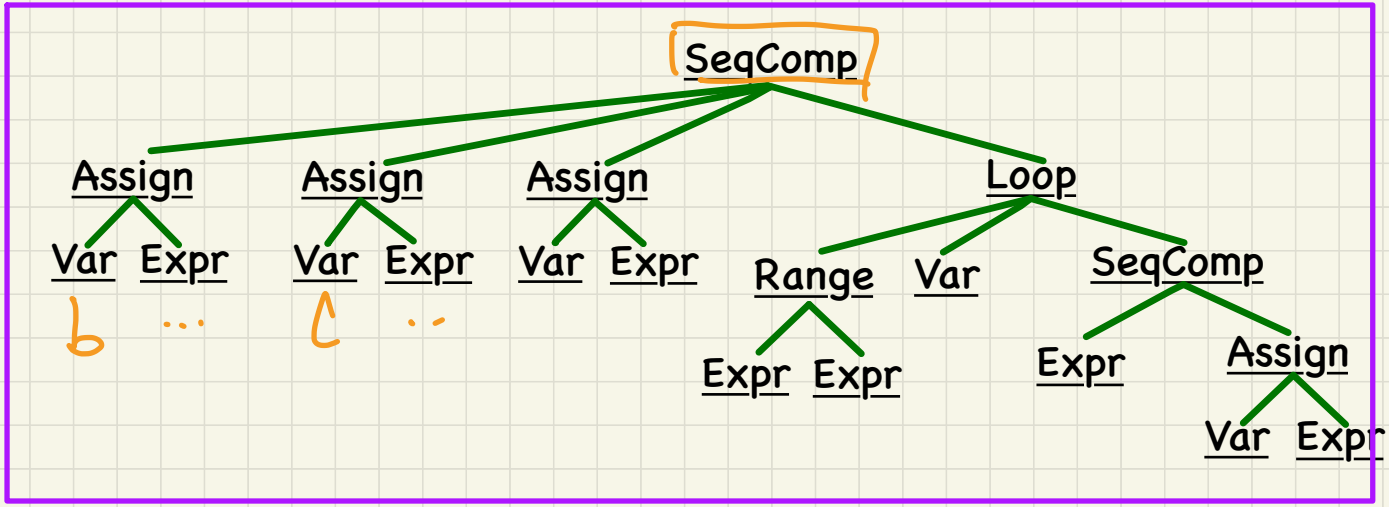
② while (true) { int i = 3; }

③ while (true) { int i = 3; int i = 4; }

Compiler Infrastructure: AST-to-AST Optimizer (1)

```
b := ... ; c := ... ; a := ...
across i |..| n is i
  loop
  read d
  a := a * 2 * b * c * d
  end
```

AST of input program:



Compiler Infrastructure: AST-to-AST Optimizer (2)

```
b := ... ; c := ... ; a := ...
temp := 2 * b * c
across i |..| n is i
  loop
    read d
    a := a * temp * d
  end
```

→ optimized version

AST of output program:

Compiler Infrastructure: AST-to-AST Optimizer (3)

Q. How should the various artifacts be connected?

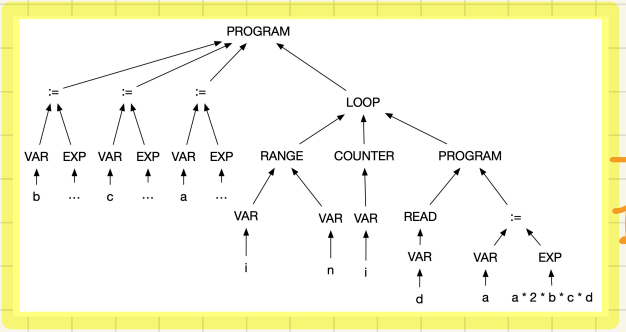
```
b := ... ; c := ... ; a := ...
across i |..| n is i
  loop
    read d
    a := a * 2 * b * c * d
  end
```

input

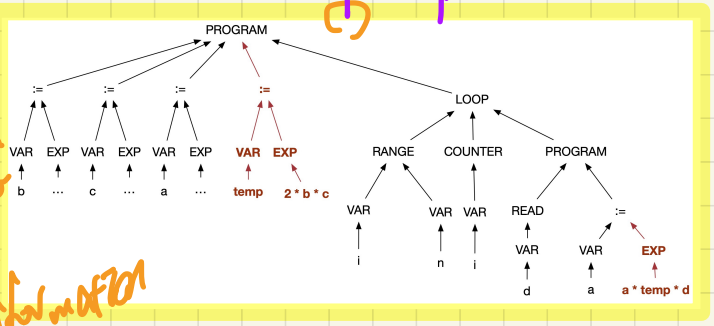
```
b := ... ; c := ... ; a := ...
temp := 2 * b * c
across i |..| n is i
  loop
    read d
    a := a * temp * d
  end
```

output

parse



IR-to-IR transformation



pretty print