COSE312: Compilers

Lecture 1 — Overview of Compilers

Hakjoo Oh
2026 Spring

What is Compiler?

Software that translates a program written in one language ("source language") into a program in another language ("target language"):

What is Compiler?

Software that translates a program written in one language ("source language") into a program in another language ("target language"):

Typically,

- the source language is a high-level language, e.g., C
- the target language is a machine language, e.g., x86

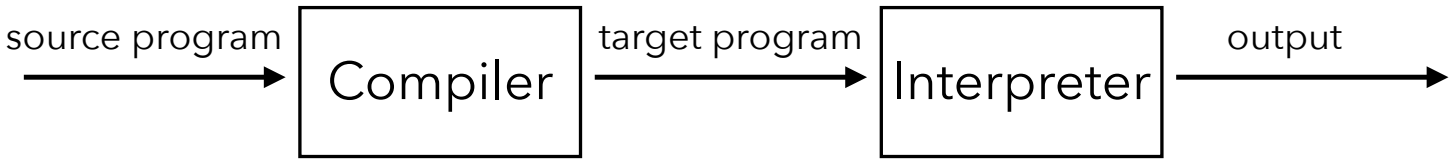
What is Compiler?

Software that translates a program written in one language ("source language") into a program in another language ("target language"):

Typically,

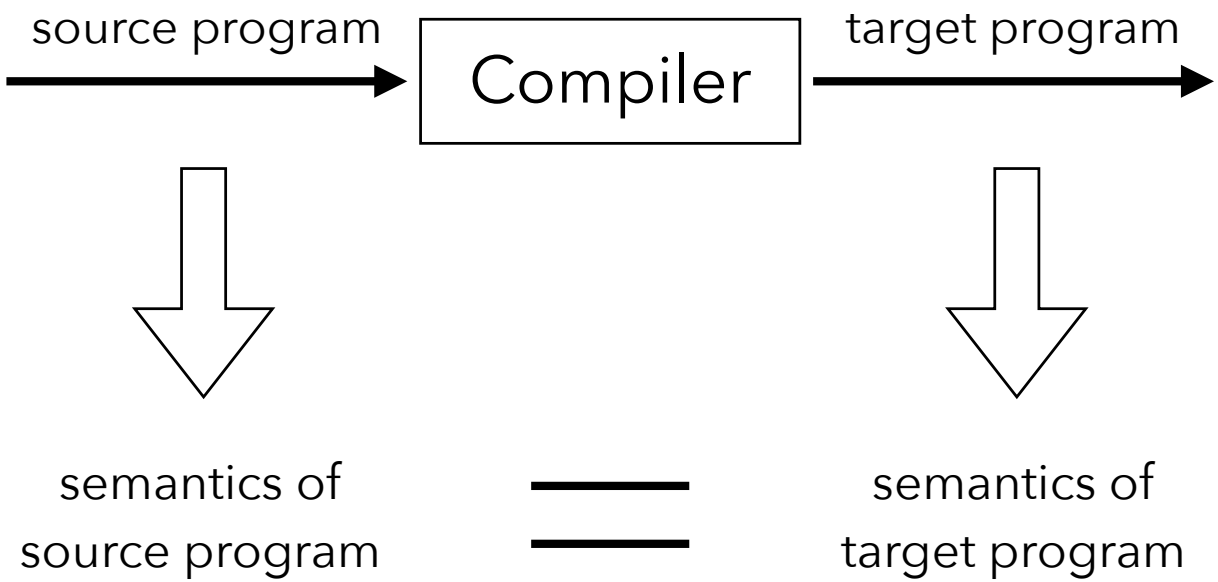
- the source language is a high-level language, e.g., C
- the target language is a machine language, e.g., x86

The target language may not be a machine language:



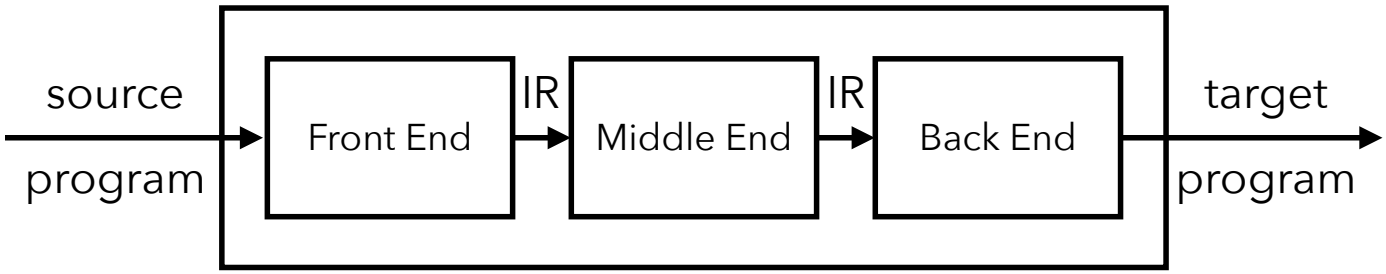
A Fundamental Requirement

- The compiler must preserve the meaning of the source program.



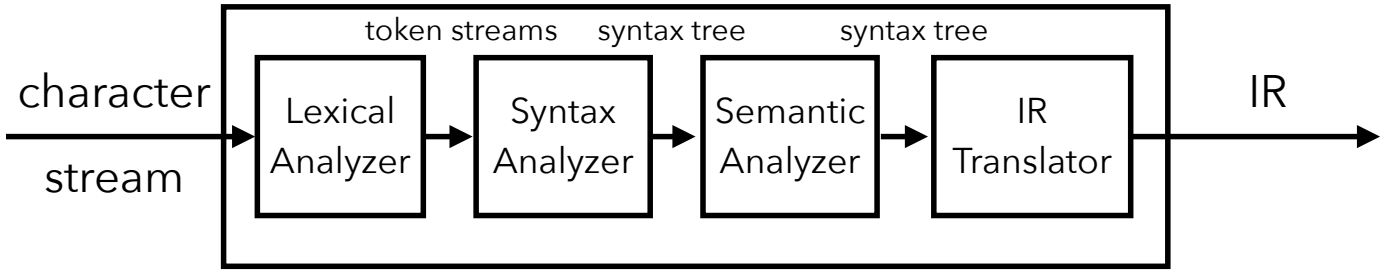
- Correctness of real-world compilers?

Structure of Compilers



- The front-end understands the source program and translates it to an intermediate representation (IR).
- The middle-end takes a program in IR and optimizes it in terms of efficiency, energy consumption, and so on.
- The back-end transforms the IR program into machine-code.

Front End



- Lexical analyzer transforms character streams into token streams.
- Syntax analyzer transforms token streams into syntax trees.
- Semantic analyzer checks the correctness of input programs.
- IR translator converts syntax trees into IRs.

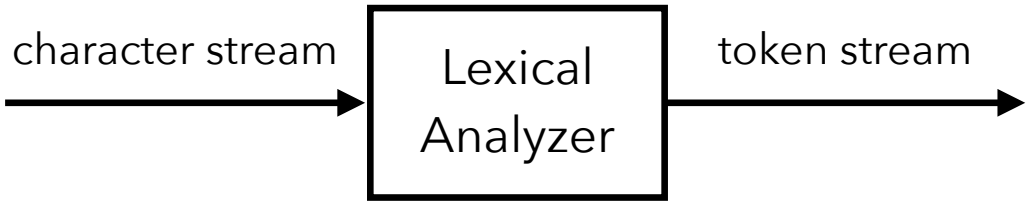
Lexical[1] Analyzer (Lexer)

A lexer analyzes the lexical structure of the source program:

[1] of or relating to words or the vocabulary of a language as distinguished from its grammar and construction

Lexical[1] Analyzer (Lexer)

A lexer analyzes the lexical structure of the source program:



ex) The lexical analyzer for C transforms the character stream

```
pos = init + rate * 10
```

[1] of or relating to words or the vocabulary of a language as distinguished from its grammar and construction

Lexical[1] Analyzer (Lexer)

A lexer analyzes the lexical structure of the source program:

ex) The lexical analyzer for C transforms the character stream

```
pos = init + rate * 10
```

into the following sequence:

"pos", "=", "init", "+", "rate", "*", "10"

[1] of or relating to words or the vocabulary of a language as distinguished from its grammar and construction

Lexical[1] Analyzer (Lexer)

A lexer analyzes the lexical structure of the source program:

ex) The lexical analyzer for C transforms the character stream

```
pos = init + rate * 10
```

into the following sequence:

"`pos`", "`=`", "`init`", "`+`", "`rate`", "`*`", "`10`"

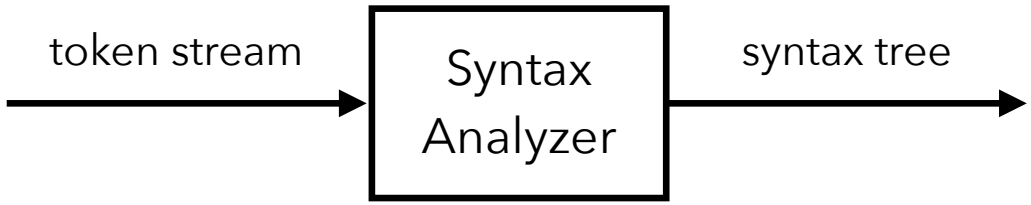
and then produces a *token* sequence:

(ID, `pos`), ASSIGN, (ID, `init`), PLUS, (ID, `rate`), MULT, (NUM,10)

[1] of or relating to words or the vocabulary of a language as distinguished from its grammar and construction

Syntax[2] Analyzer (Parser)

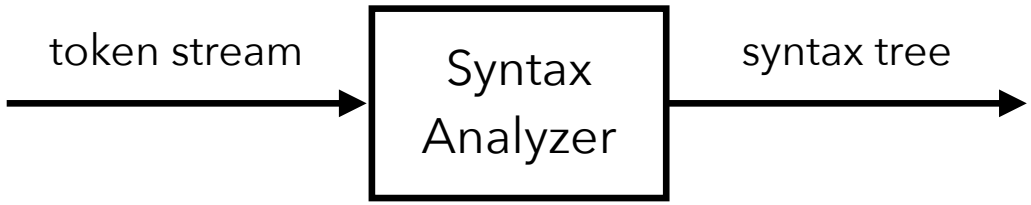
A parser analyzes the grammatical structure of the source program:



[2] the way in which words are put together to form phrases, clauses, or sentences

Syntax[2] Analyzer (Parser)

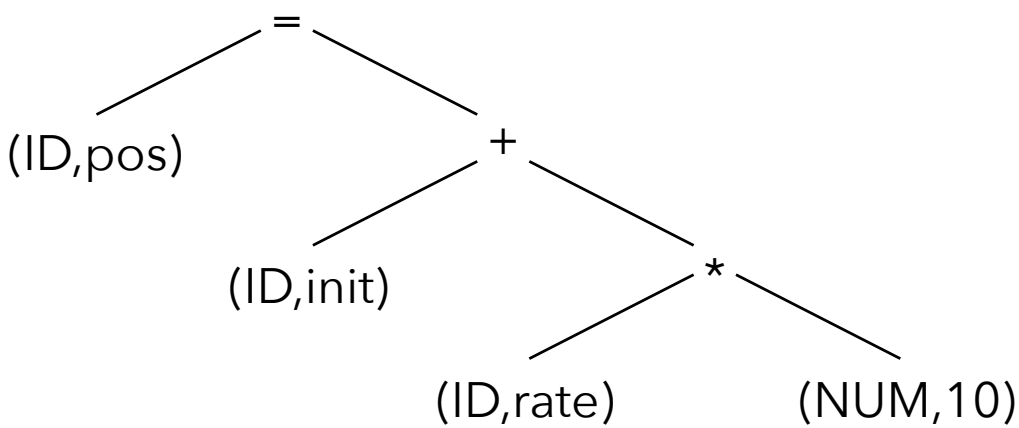
A parser analyzes the grammatical structure of the source program:



ex) A C parser transforms the sequence of tokens

(ID, `pos`), =, (ID, `init`), +, (ID, `rate`), *, (NUM,10)

into the syntax tree:



[2]the way in which words are put together to form phrases, clauses, or sentences

Semantic Analyzer

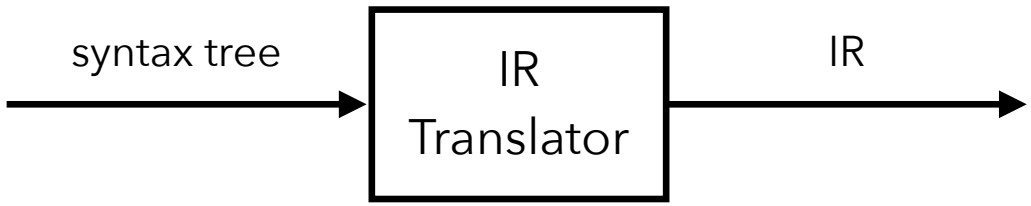
A semantic analyzer aims to establish the correctness of the input program:

- Safety errors
  - type errors, e.g.,

```
int x = 1;
string y = "hello";
int z = x + y;
```

  - memory errors (array out of bounds, null-dereference, etc)
- Functional errors
  - pre/post conditions

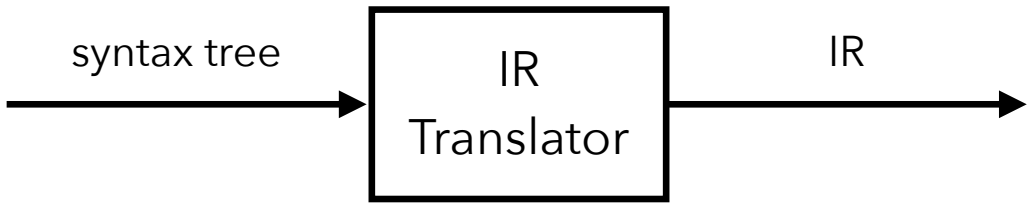
IR Translator



Intermediate Representation:

- lower-level than the source language
- higher-level than the target language

IR Translator



Intermediate Representation:

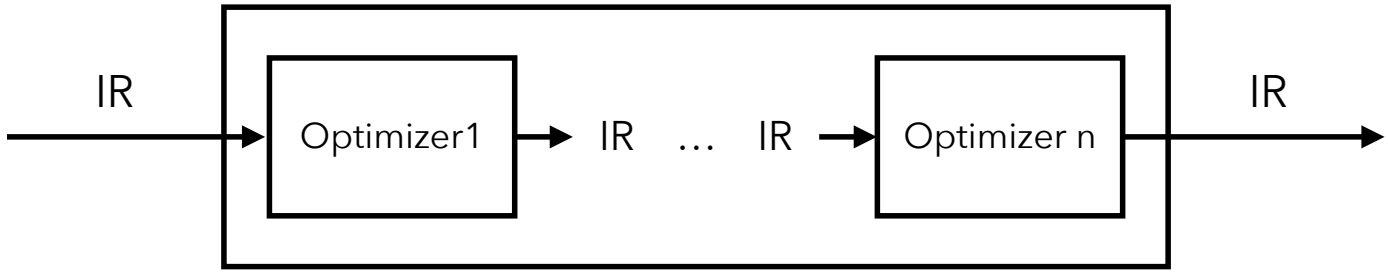
- lower-level than the source language
- higher-level than the target language

ex) Three-address code:

```
t1 = 10
t2 = rate * t1
t3 = init + t2
pos = t3
```

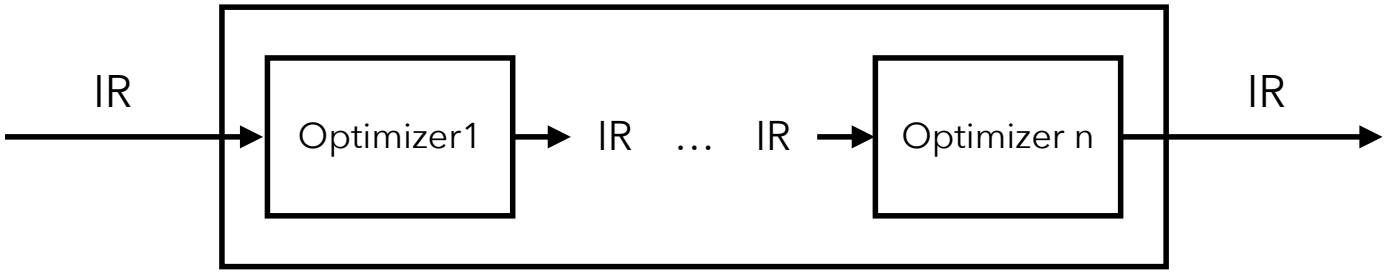
Optimizer

Transforms IR to have better performance:



Optimizer

Transforms IR to have better performance:



E.g.,

```
t1 = 10
t2 = rate * t1
t3 = init + t2
pos = t3
```

original IR

```
t1 = 10
t2 = rate * 10
t3 = init + t2
pos = t3
```

```
t2 = rate * 10
t3 = init + t2
pos = t3
```

```
t2 = rate * 10
pos = init + t2
```

final IR

Back End

Generates the target machine code:

ex) From the IR

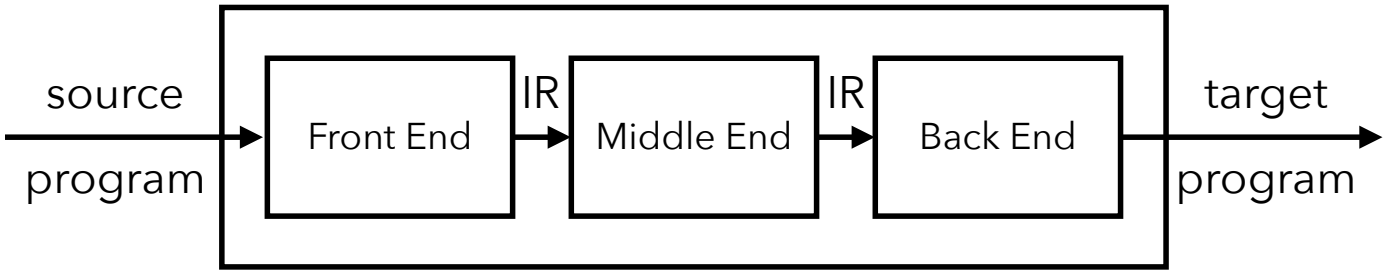
```
t2 = rate * 10
pos = init + t2
```

generates the machine code

```
LOAD  R2, rate
MUL   R2, R2, #10
LOAD  R1, init
ADD   R1, R1, R2
STORE pos, R1
```

Summary

Compilers consist of three phases:



- Front end understands the syntax and semantics of source program.
- Middle end improves the efficiency of the program.
- Back end generates the target program.