COSE312: Compilers

Lecture 1 — Overview of Compilers

Hakjoo Oh
2015 Fall

What is Compiler?

Software systems that translate a program written in one language ("source language") into a program written in another language ("target language").

What is Compiler?

Software systems that translate a program written in one language ("source language") into a program written in another language ("target language").

Typically,

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

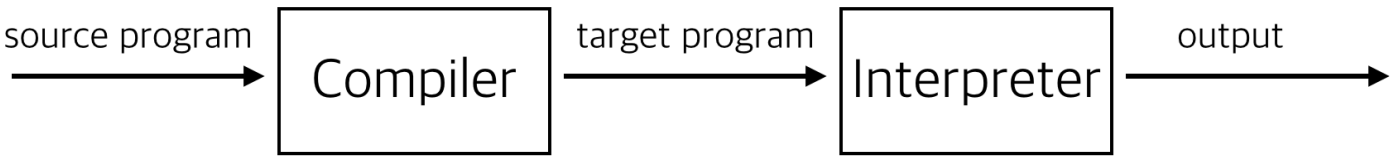
What is Compiler?

Software systems that translate a program written in one language ("source language") into a program written in another language ("target language").

Typically,

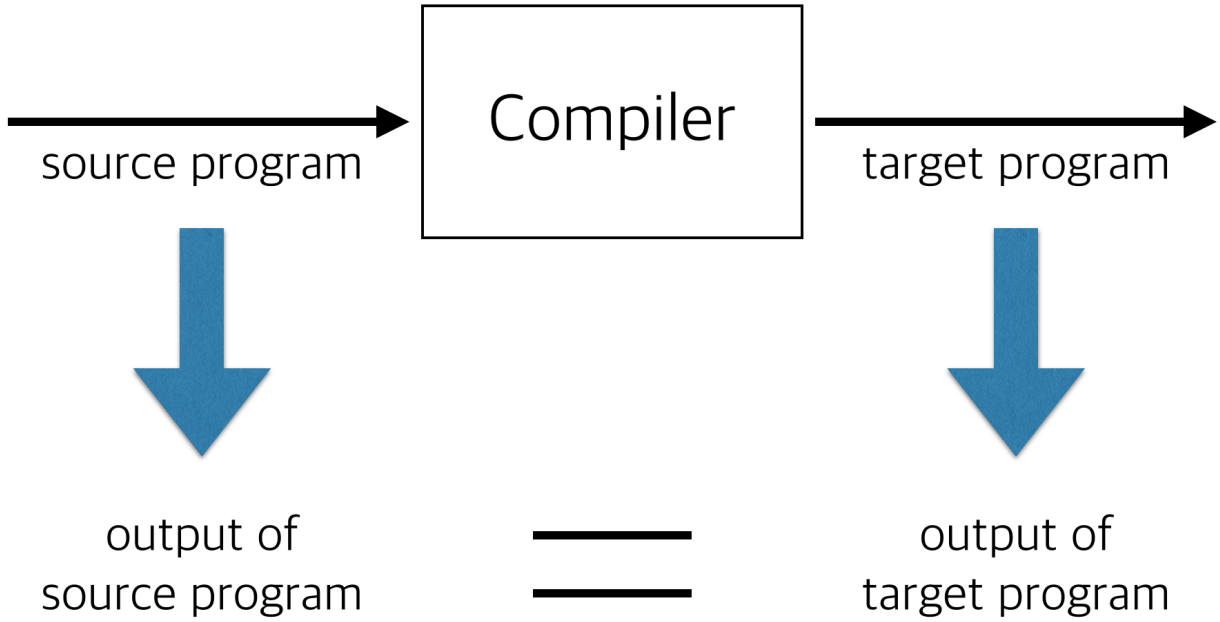
- the source language is a high-level language, e.g., C , and
- the target language is a machine language, e.g., x86.

cf) When the target language is not a machine language:

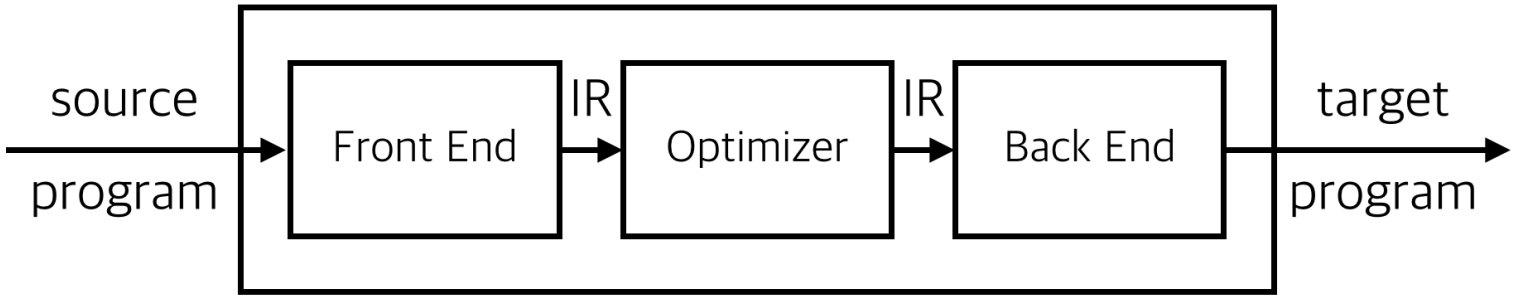


A Fundamental Requirement

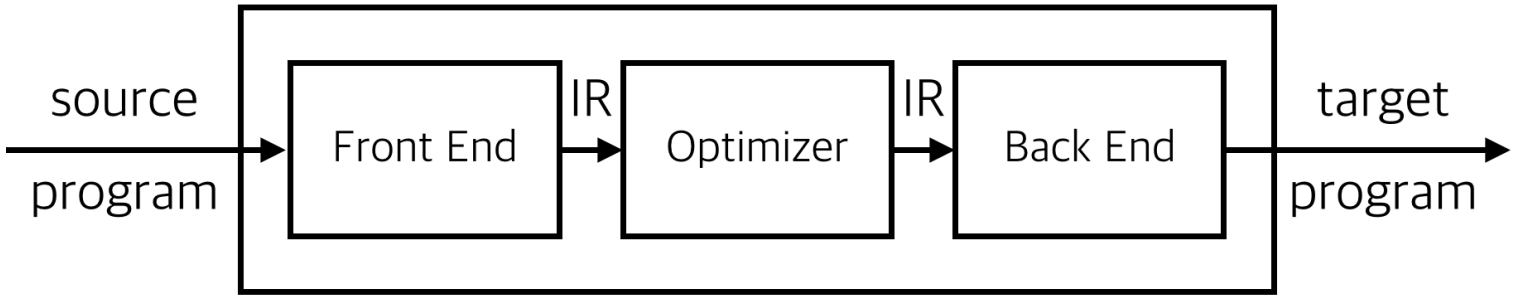
The compiler must preserve the meaning of the source program.



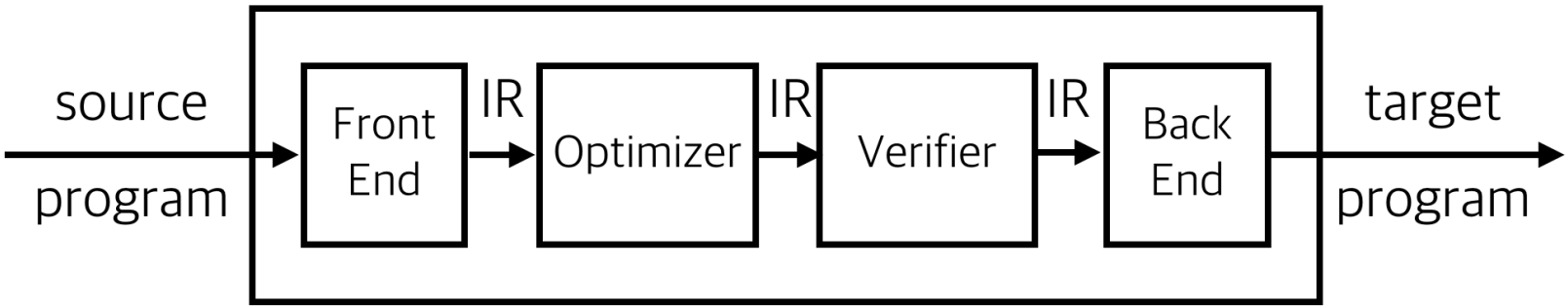
Structure of Modern Compilers



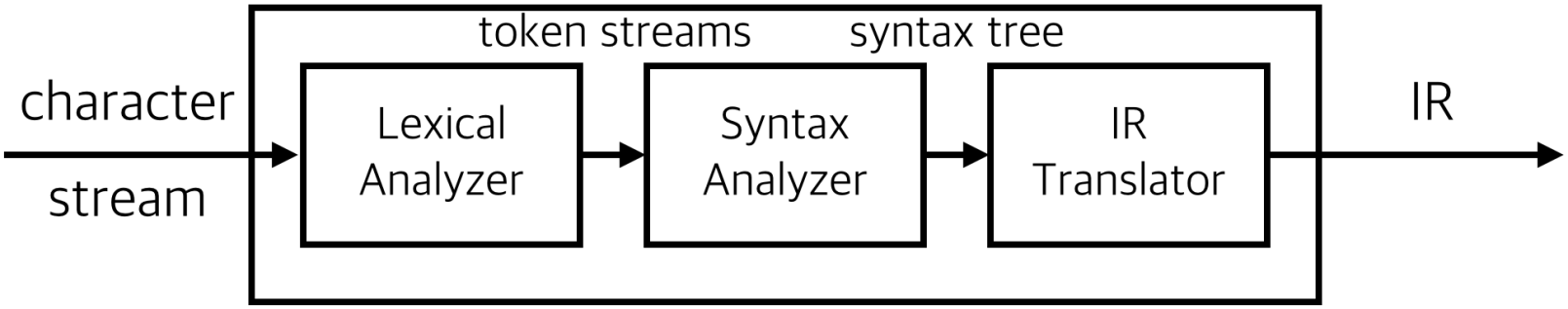
Structure of Modern Compilers



cf) "verifying compilers" (a grand challenge in CS):



Front End


token streams
syntax tree
character
stream
Lexical
Analyzer
Syntax
Analyzer
IR
Translator
IR

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 transform 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 transform the character stream

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

into a sequence of *lexemes*:

"`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 transform the character stream

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

into a sequence of *lexemes*:

"`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

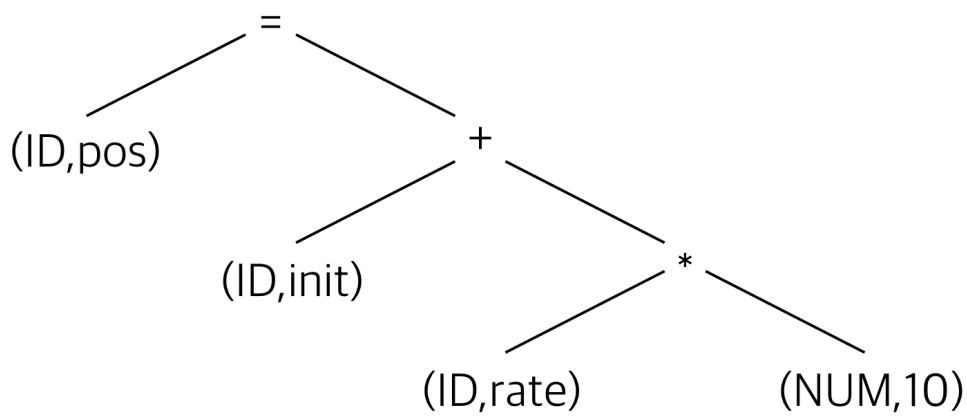
Syntax[2] Analyzer (Parser)

A parser analyzes the grammatical structure of the source program:

ex) the 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

IR Translator

Intermediate Representation:

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

IR Translator

Intermediate Representation:

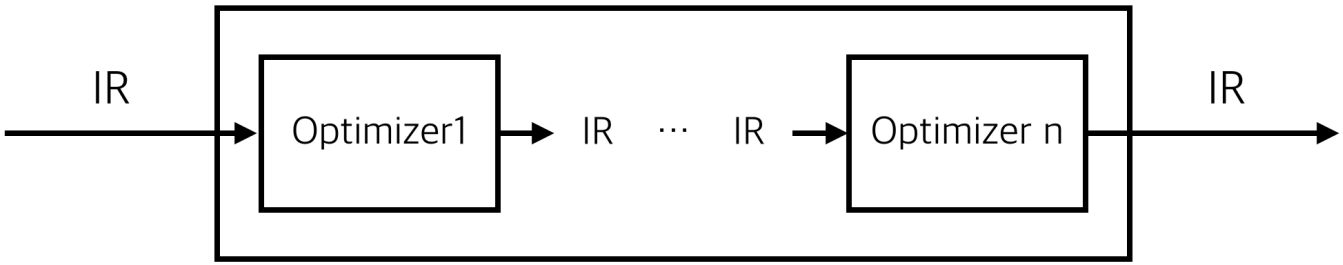
- lower-level than the source language
- higher-level than the target language

ex) translate the syntax tree into *three-address code*:

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

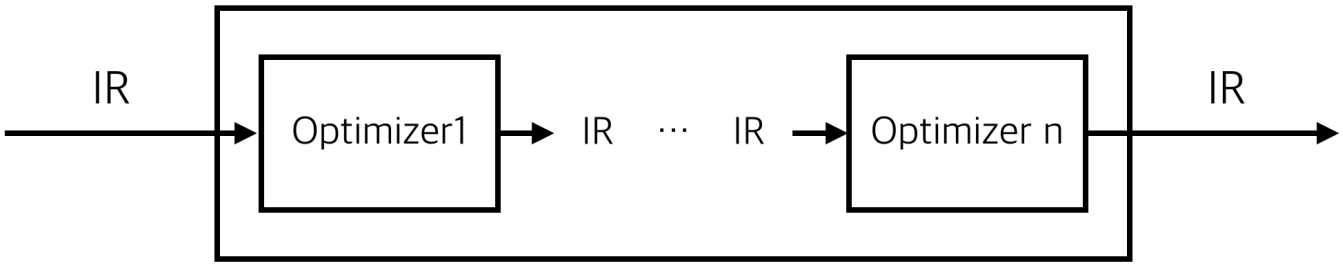
Optimizer

Transform IR to have better performance:



Optimizer

Transform IR to have better performance:



ex)

```
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

Generate the target machine code:

ex) from the IR

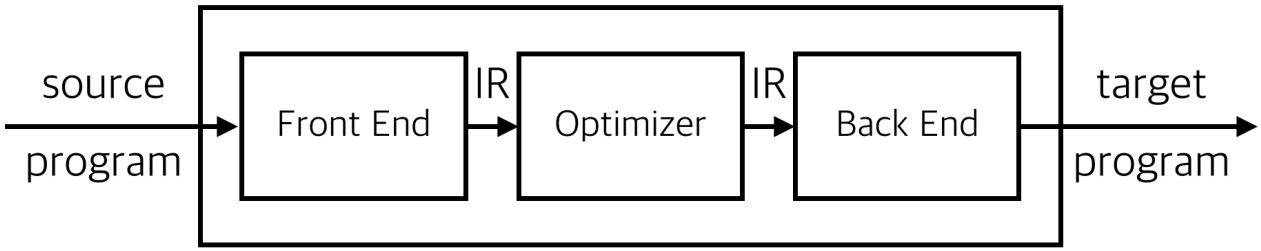
```
t2 = rate * 10
pos = init + t2
```

generate the machine code

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

Summary

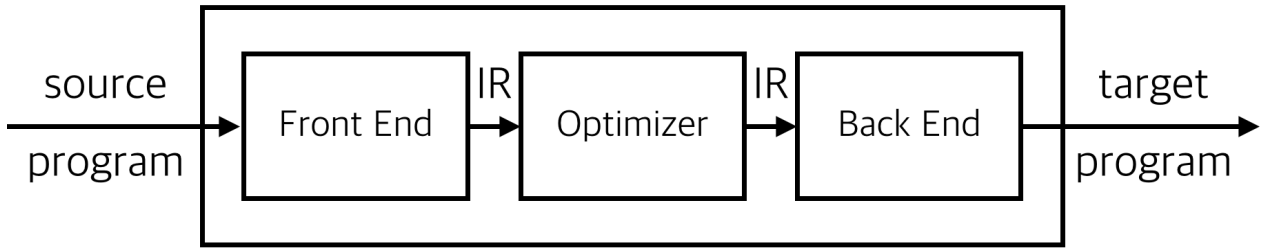
A modern compiler consists of three phases:



- The front end understands source program.
- The optimizer improves the quality of the program.
- The back end generates the target program.

Summary

A modern compiler consists of three phases:



- The front end understands source program.
- The optimizer improves the quality of the program.
- The back end generates the target program.

cf) Remember: