

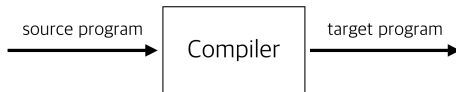
COSE312: Compilers

Lecture 1 — Overview of Compilers

Hakjoo Oh
2023 Spring

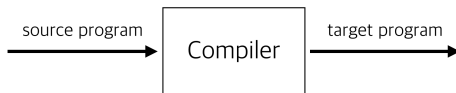
What is Compiler?

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”).

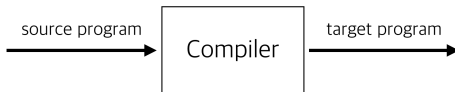


Typically,

- 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?

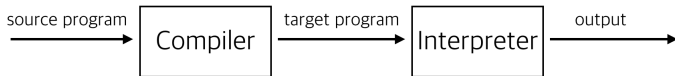
Software systems that translate a program written in one language (“source language”) into a program written in another language (“target language”).



Typically,

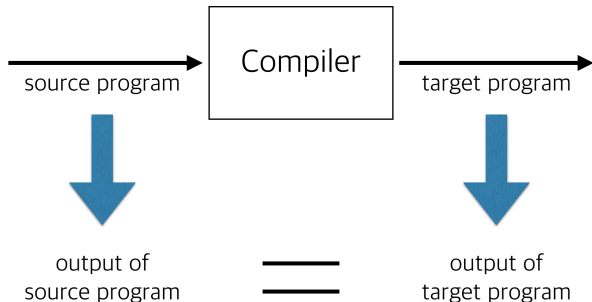
- the source language is a high-level language, e.g., C , and
- the target language is a machine language, e.g., x86.

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

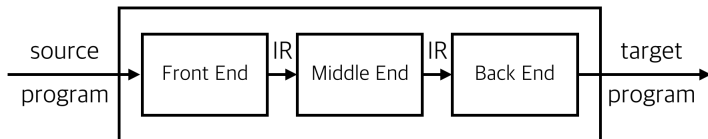


A Fundamental Requirement

The compiler must preserve the meaning of the source program.

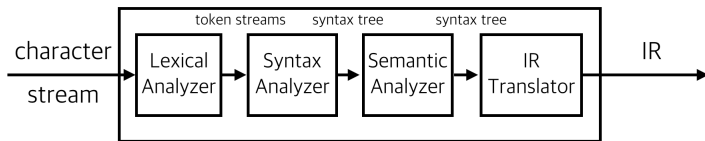


Structure of Modern Compilers



- 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.

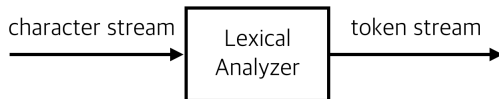
Front End



- The lexical analyzer transforms the character stream into a stream of tokens.
- The syntax analyzer transforms the stream of tokens into a syntax tree.
- The semantic analyzer checks if the program is semantically well-formed.
- The IR translator translates the syntax tree into IR.

Lexical¹ Analyzer (Lexer)

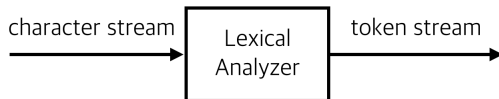
A lexer analyzes the lexical structure of the source program:



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

Lexical¹ Analyzer (Lexer)

A lexer analyzes the lexical structure of the source program:



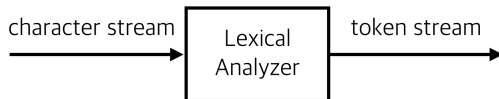
ex) The lexical analyzer transform the character stream

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

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

Lexical¹ Analyzer (Lexer)

A lexer analyzes the lexical structure of the source program:



ex) The lexical analyzer transform the character stream

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

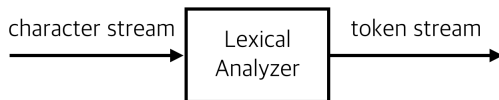
into a sequence of *lexemes*:

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

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

Lexical¹ 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"
```

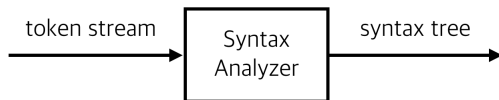
and then produces a *token* sequence:

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

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

Syntax² Analyzer (Parser)

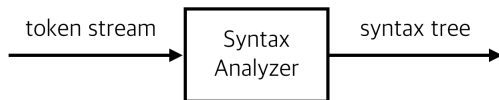
A parser analyzes the grammatical structure of the source program:



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

Syntax² Analyzer (Parser)

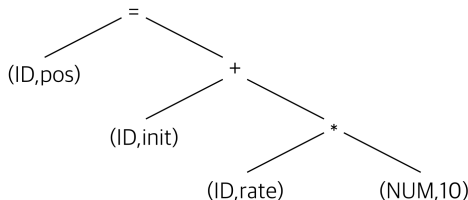
A parser analyzes the grammatical structure of the source program:



ex) the parser transforms the sequence of tokens

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

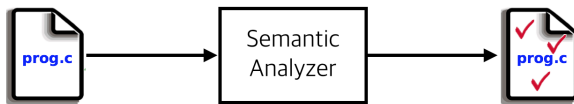
into the syntax tree:



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

Semantic Analyzer

A semantic analyzer detects semantically ill-formed programs:



ex) Type errors:

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

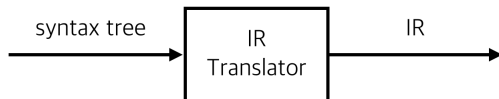
Other semantic errors:

- array out of bounds
- null-dereference
- divide-by-zero
- ...

Static Program Analysis

- Predict program behavior **statically** and **automatically**
 - ▶ **static**: by analyzing program text, before run/ship/embed
 - ▶ **automatic**: sw is analyzed by sw (“static analyzer”)
- Applications
 - ▶ **bug-finding**: e.g., runtime failures of programs
 - ▶ **security**: e.g., is this app malicious or benign?
 - ▶ **verification**: e.g., does the program meet its specification?
 - ▶ **optimization**: e.g., automatic parallelization

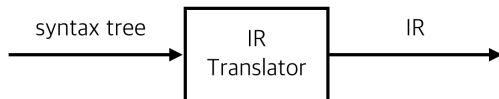
IR Translator



Intermediate Representation:

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

IR Translator



Intermediate Representation:

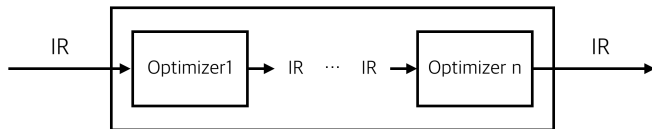
- lower-level than the source language
- higher-level than the target language

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

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

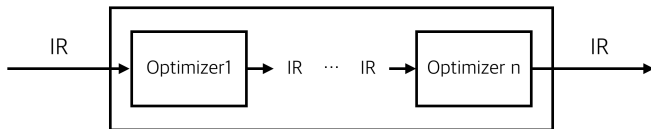
Optimizer

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
```

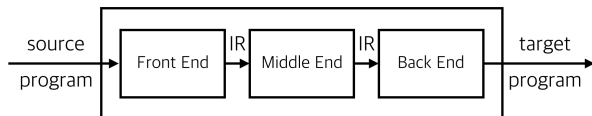
```
LOAD  R1, init
```

```
ADD   R1, R1, R2
```

```
STORE pos, R1
```

Summary

A modern compiler consists of three phases:



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